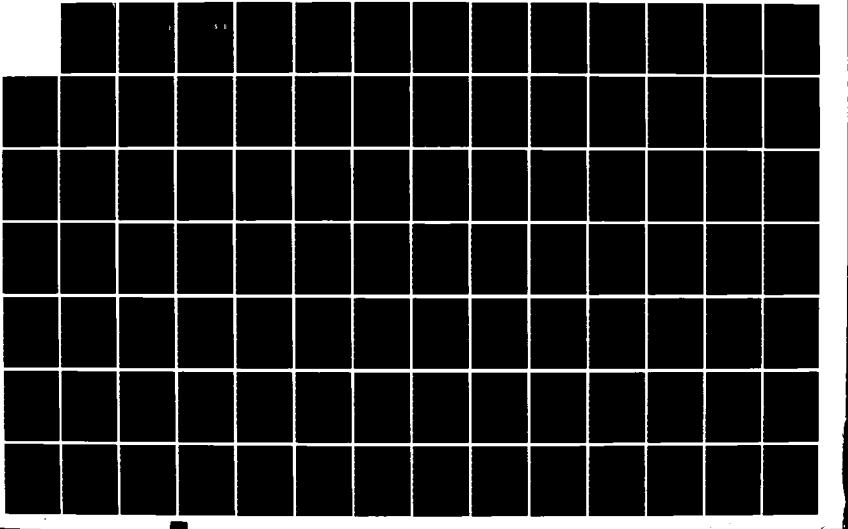
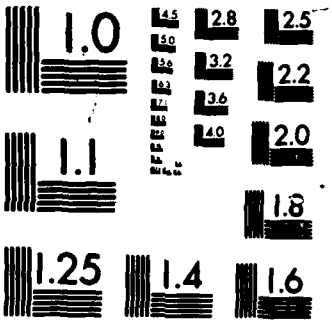
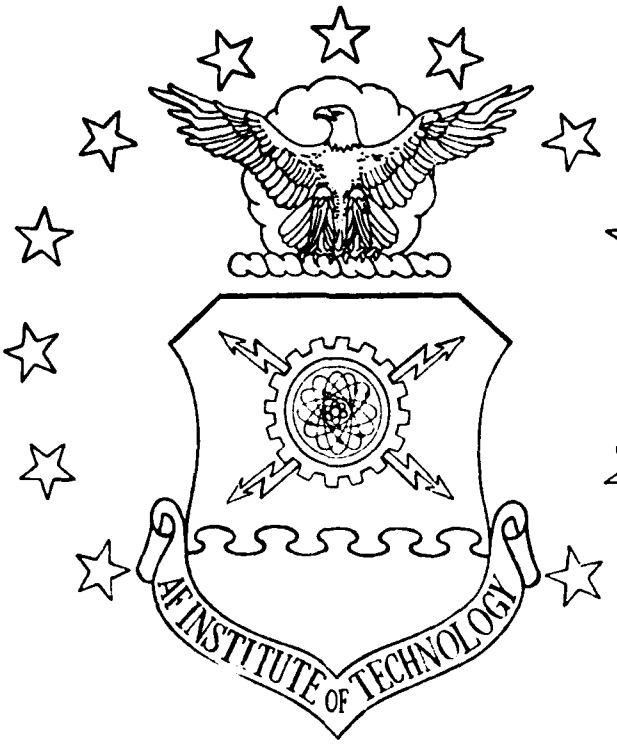
AD-A163 836 PORTABILITY EVALUATION OF AFIT-GKS AN ADA IMPLEMENTATION OF THE GRAPHICAL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/2

UNCLASSIFIED K E LEINBACH DEC 85 AFIT/GSO/MA/85D-4 F/G 9/2 NL







PORTABILITY EVALUATION OF AFIT_GKS: AN ADA IMPLEMENTATION OF THE GRAPHICAL KERNEL SYSTEM

THESIS

Kevin E. Leinbach
Captain, USAF

AFIT/GSO/MA/85D-4

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GSO/MA/85

PORTABILITY EVALUATION OF AFIT_GKS: AN ADA IMPLEMENTATION OF THE GRAPHICAL KERNEL SYSTEM

THESIS

Kevin E. Leinbach
Captain, USAF

AFIT/GSO/MA/85D-4

AFIT/GSO/MA/85D-4

PORTABILITY EVALUATION OF AFIT_GKS: AN ADA IMPLEMENTATION OF THE GRAPHICAL KERNEL SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Space Operations

Kevin E. Leinbach, B.S.
Captain, USAF

December 1985

Approved for public release; distribution unlimited

Acknowledgements

Many individuals and organizations provided a considerable amount of information and assistance to me, and I would like to acknowledge several of them here. I am deeply grateful to Professor Charles W. Richard who, as an instructor, introduced me to the exciting world of computer systems and computer graphics, and as a thesis advisor, spent countless hours poring over computer terminals and listings with me. The same thanks are extended to Lt Col Richard R. Gross who, when all seemed lost, provided the impetus and guidance to seeing this project through to the end.

I would also like to thank the Harris Corporation for their help in providing me with some of the documents required for this project. Thanks are due also to Mr. W. Nicholas Miller, AFIT's Technical Support Specialist, for the many hours he spent making the machines work, and Lt Col Joseph L. Faix for providing much needed encouragement. Finally, my thanks to Cherokee, Michael, and Valerie for providing the reason for it all.

Kevin E. Leinbach

Table of Contents

Abstract

The purposes of this ~~study~~ thesis were to develop an Ada implementation of the American National Standard Institute's new standard for computer graphics, the Graphical Kernel System (GKS), and to have it operating on an Air Force Institute of Technology (AFIT) computer system. The basis for this implementation was AFIT_GKS, a subset of GKS designed and coded by 2Lt R. Scott Ruegg on the Aeronautical System Division's Rolm Data General Computer for his 1984 AFIT thesis, AFIT_GKS -- A GKS Implementation in the Ada Programming Language. It seemed possible to install AFIT_GKS on an AFIT computer, to improve AFIT_GKS by modifying it to address more capable display terminals, and to initiate testing and evaluation of AFIT_GKS in accordance with the methods and guidelines currently under study by the National Bureau of Standards.

Considerable difficulty in porting AFIT_GKS to AFIT's computer was encountered. Some of these difficulties were caused by problems with AFIT's compiler itself, and others by the substantial changes to the Ada binding to GKS. As a result, no device interfacing or evaluation work could be accomplished. However, a framework for development of GKS in Ada at AFIT was implemented, debugged, tested, and documented, and this framework can serve well as the foundation for continued work on this subject.

PORTABILITY EVALUATION OF AFIT_GKS: AN ADA IMPLEMENTATION OF THE GRAPHICAL KERNEL SYSTEM

I. Introduction

Background Information

The Graphical Kernel System (GKS) is an internationally-recognized standard for producing computer graphics. Much of its early design work was accomplished at the Workshop on Graphics Standards Methodology in Seillac, France, in May 1976. The West German Standardization Institute, DIN, actually developed GKS in 1978. Further refinement was performed during 1980-1982 by Working Group 2 of the Subcommittee on Programming Languages of the Technical Committee on Information Processing of the International Standards Organization (ISO) (ANSI, 1984:ii).

The American National Standards Institute (ANSI) made minor changes to this international GKS and distributed a draft standard to the graphics community for review and comment in February 1984 (Ruegg, 1984:1.2). In June 1985 the Institute approved it as American National Standard GKS (ANS GKS).

Any mention of GKS in this thesis, unless specifically identified as ISO/GKS, refers to American National Standard GKS.

GKS has been chosen as a standard for three main reasons: to provide a graphics system that is easily transportable between different computer installations, to help the application programmer understand and use computer graphics, and to serve as a guideline to manufacturers of graphics equipment when they include useful combinations of graphics capabilities in new devices (Enderle and others, 1984:8).

The second major actor in this thesis is the Ada programming language. Ada, although not then known by that name, had roots as far back as the early 1970's, when the Department of Defense (DOD) became concerned over the rapidly rising cost of software for its computer systems (Booch, 1983:11). At that time, there were from 450 to 1500 different high-order and assembly languages in use in DOD systems, effectively preventing any transfer from one system to another (Booch, 1983:12). This huge number of incompatible languages often tied a project to one particular vendor or an obsolete technology. In many cases, projects were implemented in a language wholly unsuitable for the application (Booch, 1983:13).

As a result, in January 1975, the Director of Defense Research and Engineering ordered the formation of a High Order Language Working Group (HOLWG). The HOLWG was composed of members from each of the services, members of other DOD agencies, and representatives from the United Kingdom, West Germany, and France (Booch, 1983:14).

The HOLWG's work produced the language known now as Ada,

named in honor of Augusta Ada Lovelace, a nineteenth century mathematician who is considered the world's first programmer (Booch, 1983:18). Ada became a trademark of the DOD in January 1981, and it was standardized by the American National Standards Institute in February 1983 (Booch, 1983:21).

It was only fitting that these two standards, GKS and Ada, be brought together. One of the first attempts at this was by 2Lt R. Scott Ruegg in his 1984 Air Force Institute of Technology (AFIT) thesis (Ruegg, 1984). The result of his work was AFIT_GKS, a partial implementation of GKS in the Ada language.

AFIT_GKS, as an implementation of GKS in Ada, sought to exploit the concepts of portability and information hiding. The requirement for portability was a part of the design specifications for both GKS and Ada. For GKS, "portability" means the ability to use and take advantage of any type of input/output device; for Ada, it means the ability to compile and execute programs at any computer installation without forcing the user to rewrite parts of the code.

In GKS, portability is achieved by defining its actual data types (e.g., records, arrays) and their corresponding functions in abstract terms (Enderle and others, 1984:448). This abstract definition allows the principles of GKS to be used in applications written in almost any programming language.

Before a particular application program can be written in a particular language, however, the GKS data types and

functions must be mapped, one by one, into actual data types and functions available in the chosen language. This process is called the binding of GKS to a language. In binding GKS to a particular language, one is able to take advantage of (and is, in fact, restricted to) any of the facilities or constructs available in that language. For example, the GKS data type RECORD has no similar construct in FORTRAN, and it has to be simulated using arrays (Enderle and others, 1984:451). Pascal, though, does have a powerful RECORD data type, and a binding of GKS to it can take advantage of that. GKS, then, is designed so that it can use the best and most powerful facilities a particular language has to offer.

The Ada programming language provides not only a rich set of data types, but also introduces the concept of program units -- namely, subprograms, tasks, and packages (Booch, 1983:47). These program units can be used to hide information, that is, to allow an Ada program to be written so that the actual data types and procedures used for the solution of the application are hidden from the user, preventing him from using them incorrectly or trying to alter them.

It is these principles of portability and information hiding, then, that are brought together in AFIT_GKS, and any changes or enhancements made to AFIT_GKS must be made with these principles in mind.

Statement of the Problem

One of GKS's primary design goals was to provide the

entire range of modern graphics output devices such as plotters, storage tube displays, refresh displays, and color displays. There are two principles very closely associated with this goal (ANSI, 1984:2):

1) Device independence: GKS functions shall be designed to allow an application program to address quite different graphics input and output devices without modification of the program;

2) Device richness: capabilities built into a particular input or output device shall be usable by GKS.

Ruegg's AFIT_GKS provides only a basic operating capability of a full GKS implementation and does not fully realize GKS's graphics device principles. Furthermore, modern graphics devices are built with a large array of graphics utilities of the type GKS was designed to use, such as hardware implementations of clipping algorithms and much larger color and pattern possibilities. AFIT_GKS does not currently address any such utilities.

Finally, no testing or validation of any kind was performed on Ruegg's design. For a GKS implementation to be considered valid, it must be tested to insure that it does, in fact, adhere to the standard.

With these considerations in mind, this research wa conducted to answer the following questions:

-- What changes or enhancements must be made to AFIT_GKS in its present form to take full advantage of the graphics capabilities built into modern input and output devices?

-- What problems in adhering to the principles of GKS occur when Ada is the implementing language?

-- How well can AFIT_GKS stand up to testing and evaluation as the implementation of a computer graphics standard?

Before these questions could be answered, some other questions required consideration first. For example, AFIT_GKS was originally written using a DOD-validated compiler on ASD's Rolm Data General computer system. Since then, AFIT acquired its own DOD-validated Ada compiler, the Verdix VADS (Verdix Ada Development System) installed on AFIT's Academic Support Computer, a VAX-11/785 system running UNIX 4.2bsd. This compiler change promised (and in fact created) problems when Ruegg's code was recompiled on the new AFIT system.

Another question stemmed from the fact that AFIT's Raster Technologies Model 1 color display had many built-in graphics capabilities that are not currently addressed by AFIT_GKS. Could Ruegg's code handle these without extensive modification?

Finally, no testing or validation of AFIT_GKS had been accomplished. Recently, the National Bureau of Standards (NBS) had considered ways in which future implementations of GKS could be tested and certified. What progress might be made towards AFIT_GKS certification with recent NBS information?

All of these questions, then, combined to form the problem addressed by this research.

Review of Literature

Because both GKS and Ada are relatively new additions to the computer sciences, literature on the intersection of the two is quite limited. Further, because of the international origins of GKS and Ada, much of the existing literature has been published overseas. Some of the literature has just recently been standardized or is still in the standardization process.

The principal source of information on GKS is, of course, the standard itself. GKS was made an American National Standard on 17 June 1985, and the final version has not yet been published. The December 1984 draft form is available, and only editorial comments were made to it to produce the final version of the standard. The book Computer Graphics Programming (Enderle and others, 1984), although written exclusively about the International Standards Organization's (ISO's) GKS, contains information that is directly applicable to American National Standard (ANS) GKS.

The foremost source of information on the Ada programming language is the DOD's Ada Language Reference Manual, ANSI/MIL-STD-1815A, Military Standard Ada Programming Language (DOD, 1983). Many other texts and references for Ada syntax and programming techniques are available (e.g., (Barnes, 1982), (Booch, 1983).

The Harris Corporation, of Melbourne, Florida, has produced a draft binding of GKS to the Ada language under contract with the DOD. This draft has already seen several

revisions. Ruegg's AFIT_GKS used the first revision, dated 20 July 1984 (X3H3/83-95r1, 1984). The next revision, considerably different from the first, was released on 1 February 1985 (X3H3/83-95r2, 1985). Information about the third and most recent revision, yet to be published, was obtained in a telephone conversation with Harris software engineer Barbara Furtney on 18 August 1985 (Furtney, 1985b). This pending revision, while leaving most of the GKS/Ada types from Revision 2 intact, does make some fairly significant changes to some of the larger utility packages, such as GKS_LIST_UTILITIES and GKS_COORDINATE_SYSTEM.

Although not yet recognized by ANSI as standard, this revised Harris binding is currently under consideration by that agency for standardization, and Harris expects this binding to be declared an ANSI standard very soon and to be included as part of the ANS GKS standard (Cuthbert, 1985). Ruegg's AFIT_GKS was written in accordance with the first version of the Harris binding, and in this research all attempts were made to retain as much of Ruegg's design and code as was possible with the latest (pending) version of the binding.

There is very little testing and evaluation information that has been published in the United States. What does exist is adapted from the ISO's literature. In 1981, shortly after the beginning of the GKS development process, work started on a closely related activity: the certification of graphics systems (Enderle and others, 1984:485). Conscious

of the fact that a standard is largely worthless unless there is a procedure to test the conformity of implementations to it, the ISO formed a special subgroup to develop a certification/validation scheme and to guide the GKS review process in its final stages (Enderle and others, 1984:485).

There is no doubt that some certification process is required. Even when GKS was in its earlier draft stages, major United States companies began developing GKS implementations, microcomputer software manufacturers were hastily offering "GKS-like" packages, and hardware manufacturers were announcing "GKS workstation capability in a terminal" (Brodlie, 1984:13). A trend like this, taking the standard as a rough guideline rather than a blueprint, would forfeit all the advantages of portability intended by GKS (Brodlie, 1984:13).

Most of the literature available on the subject of certification and validation of GKS has been written by members of the international computer graphics community. They have covered many topics, such as the different approaches to testing with their inherent advantages and disadvantages, and discussions on where GKS is best tested: at the device, application, or user interface.

The Technische Hochschule Darmstadt in West Germany has developed an initial test program suite for testing GKS at the application interface (National Bureau of Standards, 1985). This suite of programs is a preliminary test version (dated November 1984) produced even before GKS was declared

standard by the ISO. It has been made available to, and is currently under study by, the United States National Bureau of Standards (NBS), the agency responsible for developing certification procedures for ANS GKS. The NBS has, in turn, recently released this test program to US organizations and institutions involved in GKS implementation. It is distributed exactly as it was received by NBS, and therefore is offered without warranty or endorsement by NBS. NBS intends the programs to be used by the individual implementors to obtain information about their implementation's adherence to GKS, to evaluate the tests with respect to correctness, design philosophy, desirable enhancements, and documentation, and to provide feedback to the NBS of their evaluations (NBS, 1985:cover sheet).

Sequence of Presentation

The next chapter lays out the specific requirements for completing this research. The third chapter discusses the process of installing AFIT_GKS on the AFIT Academic Support Computer and the problems that arose during that process. Chapters IV and V discuss, respectively, the attempts at identifying and resolving these problems, and the results obtained from those attempts. Finally, Chapter VI provides some conclusions and recommendations for further study of AFIT_GKS.

II. Requirements

It was apparent at the outset that completion of this project would each require substantial effort. Installation of AFIT_GKS on the AFIT computer system and the subsequent upgrading of AFIT_GKS to enable it to address the graphics capabilities of the Raster Technologies Model One color display would require considerable software system design. Furthermore, extensive test and evaluation of the resulting product would be required to compare it to other graphics systems and devices. The three main tasks, then, required to complete this project can be summarized as follows:

-- Loading and compiling of Ruegg's source code on the AFIT Academic Support Computer system, and attaining an equivalent operating capability on this system.

-- Enabling the capabilities of the Raster Technologies Model One color display to be addressed by AFIT_GKS.

-- Designing a suitable testing and evaluation method, and applying it to learn how the rehosted AFIT_GKS performed.

Notice that completion of the second and third tasks are dependent on the completion of the first task; that is, it would be extremely difficult to do any interface work or testing of AFIT_GKS if the rehosting of AFIT_GKS was not successful.

Let us now examine the requirements of each of these three tasks in more detail.

Realize AFIT_GKS on an AFIT Computer System

At the time that Ruegg accomplished all of his original

AFIT_GKS work, there were no Ada compilers available on any of the AFIT computer systems. The only available DOD-validated compiler was located on the Rolm Data General computer system at Aeronautical Systems Division (ASD). While this system could be accessed through AFIT telecommunication lines, it was far more desirable to have AFIT_GKS maintained on an AFIT computer system once AFIT obtained its own DOD-validated Ada compiler. There were also funding and authorization problems to be dealt with when trying to use ASD computer resources.

Since all of Ruegg's source code was available on computer tape, this task was initially seen as essentially an exercise in rehosting an Ada project from one DOD-validated compiler to another. No reorganization of Ruegg's code or design was anticipated or planned; the purpose of this step of the project was solely to transfer AFIT_GKS to AFIT computer resources.

Interfacing the Raster Technologies Model One

Ruegg had two display devices working with his original AFIT_GKS implementation: the Tektronix 4014, a storage-tube type device, and the Tektronix 4027A, a raster color display. These two devices are essentially "dumb" display terminals, and they were interfaced as such in AFIT_GKS. For example, all the clipping of line segments was accomplished by software on the host machine, an implementation which can add an appreciable delay to the output on the display surface.

The Raster Technologies Model One is a newer device with capabilities that far exceed those of older display devices. It has a standard display space resolution of 512 x 512 pixels, but it can also be used in a 1024 x 1024 mode (Raster Technologies, 1983b:17). Resident in firmware in the terminal are algorithms for window and viewport definition, clipping, and transformations for rotation, translation, and scaling (Raster Technologies, 1983a:6). The upgraded version of the Model One, the Model One/25-S, has additional firmware and memory for polygon shading and hidden surface removal through the use of a 16-bit Z-buffer (Raster Technologies, 1984:3).

With the Tektronix 4014 and 4027A, the functions for defining windows and viewports, clipping, and the three kinds of transformations are all performed by software on the host machine. By moving these routines down into firmware, the Model One can significantly reduce host machine memory and processing time requirements.

The interface of the Model One to AFIT_GKS, then, should take advantage of these highly attractive features, ideally without disturbing the existing interfaces for the Tektronix 4014 and 4027A.

A further requirement is related to the preceding one. In conjunction with the development of the GKS standard was the development of the Computer Graphics Virtual Device Interface (CG-VDI) standard. This latter standard, initiated in 1980 by Technical Committee X3H3 of the ANSI, includes

elements to provide support for GKS (ANSI, 1985:iv). The objective of the CG-VDI is to provide "a standard functional and syntactical specification of the control and data exchange between device-independent graphics software and one or more device-dependent graphics device drivers" (ANSI, 1985:7), and thus it serves as an interface standard for computer graphics software package and device driver implementors (ANSI, 1985:7).

The CG-VDI standard exists currently only in draft form, and it draws extensively for its model of a graphics system on GKS (ANSI, 1985:6). Since the CG-VDI will eventually be published as an American National Standard and all new display devices and their interfaces will be expected to conform to it, it was felt that any further interfacing work accomplished on AFIT_GKS should be done in accordance with the concepts outlined in the CG-VDI.

Certification/Validation of AFIT_GKS

The test programs provided by the NBS arrive in the form of a magnetic tape containing approximately two megabytes of source code, test documentation, and user manual, some of which is still in the original German language. Written entirely in FORTRAN for testing FORTRAN implementations, the test programs are designed to test GKS operator functions at GKS level 0a and data structures at GKS level 2b. The user manual describes the procedures required to install the test programs and provides sketches showing the output expected at

each stage of the tests.

As it is designed to test only the draft version of ISO/GKS, the changes required to use this program to test an Ada implementation of ANS GKS were quite significant, requiring more than a blind translation of the code into the Ada language. However, portions of the programs are adaptable (albeit not easily), and, because the GKS certification process will eventually be performed by the NBS, any testing of AFIT_GKS should be accomplished using these NBS-provided test programs as a guideline for what items should be tested and to what depth those items should be tested.

For example, the first test program described in the manual tests for the proper opening of GKS and the subsequent opening and activation of a workstation. The routine then draws a title frame comprising two lines of text enclosed within a border. The output produced should resemble that shown in Appendix A. The manual then provides "checkpoint" questions to aid the user in evaluating how well his implementation responds to each of the tests.

Subsequent tests check for the various attributes required for a GKS implementation, such as different line styles, line widths, and marker types and sizes. For example, the second routine tests for the four mandatory linetypes: solid, dashed, dotted, and dashed-dotted. The output produced is compared with the standard (again shown in Appendix A) and evaluated with the checkpoint questions for that particular test.

Because Ruegg's AFIT_GKS does not provide many of the various text attributes mandated by GKS, the text testing routines will not produce much useful information. However, most of the basic GKS functions and the other output attributes (polyline, marker, color, etc.) have been implemented and should be tested.

III. Installation of AFIT_GKS On AFIT Computer System

Intended Procedure of Installation

At the beginning of this project, AFIT_GKS existed only in the form of a computer tape of 47 files and paper listings of a few of them. Also available were copies of Ruegg's thesis and the documents he worked from.

There were several AFIT computer systems seen as possible new host machines for AFIT_GKS:

1) VAX/UNIX 4.1bsd (Scientific Support Computer)
2) VAX/VMS (Course Support Computer)
3) VAX/UNIX 4.2bsd (Academic Support Computer)

Each of these systems had DOD-validated Ada compilers on order. However, the first compiler to be received by AFIT was the Verdix Ada Development System (VADS) for the Academic Support Computer (ASC), which arrived during the first week of June 1985 and was installed shortly thereafter. The ASC was quite satisfactory because it already had the UNIX 4.2bsd operating system installed (and would not have to be upgraded during the project, unlike the SSC); it also was the least heavily-utilized computer system at that time.

Although the AFIT_GKS tape was not in UNIX "tar" format, no problems were anticipated in loading the source code. Once the source code was loaded, identifying the files and determining the required order of compilation was to be the next step. Ruegg had provided no file name or even file

dependency information, so this information would have to be learned simply through visual inspection of the source code and the "with" and "use" statements at the beginning of each file.

Once the required file dependency information had been obtained, compilation of each of the files would follow. This was not expected to present any major problems other than, perhaps, some machine-specific library handling differences. No other problems were anticipated because both the Rolm Data General and the ASC were using DOD-validated Ada compilers.

Actual Procedure

Problems arose both during the tape loading process and the subsequent compilations of the AFIT_GKS files. Initial loading of the AFIT_GKS tape was attempted, as intended, directly on the ASC through its tape drives located in Building 641. However, the tape would not load. While this failure appeared initially to be a tape problem, investigation by the ASC system operator and the technical representative determined that the problem was caused by local tape handling and management software. These commands had not been installed properly and gave very misleading error messages.

Because of the ASC tape drive problems, a rather circuitous method had to be used to get the tape loaded. The tape had to first be dumped to the SSC VAX/UNIX 4.1 system

using UNIX's 'dd' utility, using the parameters as shown in Appendix B. These files were then transferred to the ASC VAX using the 'uucp' utility, where still another pass with the 'dd' utility (Appendix B) had to be made to insert linefeeds at the appropriate places. This solution was found only after many trial-and-error attempts, compounded by the lack of information on how the source code was originally written to the tape. These attempts cost about two days.

The next step was to examine the source code to determine the individual Ada unit names in each of the files and to establish the correct order of compilation. The files amounted to approximately two megabytes of source code. After all the files were renamed and renumbered with more descriptive yet easily-handled names (Appendix C), the file dependency was established by manually checking through listings of the files and examining the "with" and "use" statements in each file. The file dependencies thus established are shown in outline form in Appendix D. As it turned out, the files could be compiled in the order they were written to the tape, but this was not apparent from Ruegg's documentation, and in any case the dependency "tree" proved to be quite useful during the course of this project by helping to trace the formation of complex data and package structures.

Results and Problems

During the discussion that follows, files are referred to by the letter "f", a two-digit number, and an asterisk

(*). These file numbers are simply shorthand for the complete file names as shown in Appendix C.

The first attempt at producing an executable demonstration of AFIT_GKS was to get Ruegg's procedure DEMO (f46*) and its required files compiled. The files that needed to be compiled to do this were f02* through f31*, f33*, and, finally, f45*, the DEMO procedure. Of these 32 files, fourteen, or almost half, required changes to allow compilation by the VADS compiler. Almost all of the files that needed no changes were simple package specifications only. Most (twelve) of the files needing changes were package bodies. This was a surprising discovery considering that all these changes were required only to transport the source code from one DOD-validated compiler to another!

A few of these changes were obvious and made easily; however, many of them required more effort and often could not be made without some lengthy trial-and-error testing. The changes that were made easily dealt with the library differences between the implementations of Ada on the Rolm Data General and the ASC. For example, the Rolm Data General uses a library package called CORE_FUNCTIONS to supply the standard mathematical functions, a package called NUMERIC_IO to permit output of different numerical values, and a package called TTY_IO to output characters. On the VADS, the mathematical functions are supplied by the package BASIC_MATH, and integer and floating point values are provided by instantiations of the generic INTEGER_IO and FLOAT_IO packages (which

are contained in package STANDARD). These two instantiations, named "int_io" and "numeric_io", were placed in file f50* for convenience. Finally, the VADS handles normal character and control character output with package STANDARD and package ASCII (contained within package STANDARD). These differences required changes to files f02*, f08*, f10*, f14*, f16*, and f18*.

The other changes required were not nearly as simple and straightforward. The first problem appeared in the first file to be compiled: f02*, the GKS_COORDINATE_SYSTEM generic package. The line that caused this problem (in this case, an internal compiler error) was the line defining the subtype MAGNITUDE. Apparently Ruegg had some trouble with this particular type also, as shown in Appendix E, which is taken directly from Ruegg's code. As noted before, the intent was to retain as much of Ruegg's original code as possible, even though this code used the outdated Revision 1 of the Harris binding. Therefore, the best attempt at a fix was to comment out the MAGNITUDE definition that Ruegg had added, and use instead the five lines of code that were the original Revision 1 binding definitions for MAGNITUDE, in the hope that the VADS compiler would be able to handle what Ruegg said the Rolm wouldn't. This did not work at first either, because these five lines contained three typographical errors, which also appear in the Harris binding. Though Ruegg mentions that he thought the Harris definition was wrong, there was nothing to indicate that he tried correcting these three

typographical errors. Finally, changes were made to the lines as follows:

```
type MAGNITUDE_BASE_TYPE is digits MAGNITUDE_PRECISION;

subtype MAGNITUDE is MAGNITUDE_BASE_TYPE range
        MAGNITUDE_BASE_TYPE( COORDINATE' SAFE_SMALL)..
        MAGNITUDE_BASE_TYPE( ABS( COORDINATE' LAST
                                - COORDINATE' FIRST));
```

The file now compiled correctly. However, four other files required changes to allow for the newer definition of MAGNITUDE_BASE_TYPE in f02*. The newer definition also caused the operators "<" and ">" in f24* to become undefined. These problems were corrected by prefixing the offending values with the proper type conversions. This problem took approximately fifteen days to correct and was typical of the kinds of problems which seriously affected the portability of AFIT_GKS.

The next serious problem occurred during compilation of file f07*. This problem, also an internal compiler error, occurred on the following line:

```
CURRENT_HEIGHT : CHAR_HEIGHT := 1.0;
```

Examination of the definitions of the data types involved seemed to indicate that this line was completely legal, and when an internal compiler error is produced, no clue as to its cause is supplied. Trial-and-error attempts were again necessary to isolate and identify the error. This error was eventually sidestepped by rewriting the line as follows:

```
CURRENT_HEIGHT : CHAR_HEIGHT := CHAR_HEIGHT( 1.0);
```

In two other files, f10* and f16*, Ruegg had used a

local variable "i" of type INTEGER to serve as a counter in a loop statement. Such a local variable is required in Pascal, but not in Ada, and it appears that Ruegg had accidentally and momentarily switched to using the Pascal language. The VADS produced a warning message indicating that the "i" in the loop statement would hide the value of the locally declared "i". Whether the Rolm compiler supplied this same message is unknown, but it seems reasonable to believe that Ruegg would have removed the superfluous local "i" declaration if the Rolm had also given him a warning. Thus, there appears to be an additional difference between the two validated compilers: the situations that produce error messages.

In five procedures within f24*, a local dummy value holder had to be added because the VADS objected to the way Ruegg had used an OUT procedure parameter. Apparently this error had gone undetected and unchecked with the Rolm compiler.

Finally, several files had lines which were longer than 80 columns, and during the tape loading process, these lines were truncated. Fortunately, the missing source code could be reconstructed from the partial program listings left by Ruegg.

The changes enumerated above eventually enabled compilation of all the files required to run the DEMO program, and an error-free executable ("a.out") file was produced. Upon execution, however, the program immediately crashed with an Ada exception. The VADS debugger was used, and it pointed to

the definition of MAGNITUDE_BASE_TYPE back in fO2* as the cause of the problem.

At this time, the best course of action seemed to be to rewrite those portions of fO2* dealing with MAGNITUDE to the specifications given in Revision 3 of the Harris binding, while still leaving the rest of AFIT_GKS with its original Revision 1 specifications. This was accomplished, and the compilation process of the 32 files began again.

Three more internal compiler errors were encountered during this process, and once again, trial-and-error attempts were made to resolve them. This was a time-consuming process, taking about two weeks to complete. Often the internal error would not show up until compiling one of the last few files, and the change to solve that error would need to be accomplished in one of the first few files, requiring the recompilation of all of the files in between.

Again, an error-free "a.out" file was eventually produced. Upon execution this time, however, a UNIX error message "Illegal instruction (core dumped)" was received. The VADS debugger was not extremely helpful this time because the error message was generated not by Ada, but by UNIX, indicating that the problem most likely lay with the Ada/UNIX interface. At this point, over half of the time available for this project had been invested, indicating the need for a new approach to the problem. The new approach taken is discussed in Chapter IV.

Evaluation of GKS, Ada, and the Verdix VADS Compiler

Before beginning a discussion of the attempts to locate and fix the porting problem encountered with AFIT_GKS, it might be useful to evaluate the systems and tools used in this project: GKS, Ada, and the VADS compiler.

Although GKS's stated goals of portability and device richness are quite commendable, they are attainable only at the expense of another important software system design goal: simplicity. Attempting to be able to include any graphic input or output device, on any computer system, using almost any computer language, is no easy task. GKS accomplishes it, but only by defining 186 functions and well over 100 data structures. And GKS is currently only a two-dimensional graphics system; how large will it become when the third dimension is added?

Furthermore, this size renders GKS unsuitable for small graphics systems not requiring a host computer and designed for one specific purpose. Consider, for example, a hand-held personal computer with limited and fixed graphics capabilities, or perhaps even a cash register. It is infeasible to make these machines conform to GKS specifications. At present, there are certainly needs for graphics that are simply not economically satisfied by GKS, and even though GKS can be subsetted by using one of its lower levels, there are now many small-scale systems in which implementing an application the simplest and cheapest way will be more attractive than forcing it to conform to GKS.

Ada, as a fairly new high-order computer language, is drawing both criticism and applause. A typical complaint is that Ada has "tried to be all things to all people" (Munro, 1983:212). The applause comes from those who say that Ada not only encourages good design and programming practice, but enforces it (Booch, 1983:4). This project illustrates that Ada, although resembling Pascal, is certainly not as easy to learn as Pascal. Even after an Ada course, this programmer needed the Ada Language Reference Manual and two other Ada reference books at all times to accomplish anything on this project. It is, though, a very capable language, and well suited to large software systems.

The Verdix VADS used in this project is the Verdix Corporation's first release of their DOD-validated Ada compiler. In addition to the compiler, the release contains several useful tools for Ada program development: a source code debugger, library maintenance facilities, facilities to handle automatic recompilation of files following changes, and a facility for reporting problems found by the user. Overall, use of the VADS was quite enjoyable. It was adequately fast, the debugger facility was often quite helpful, and library maintenance required almost no effort on the user's part. There were problems, however, notably the internal compiler errors mentioned previously. These are not surprising for the first release of a new compiler. The product support technicians at Verdix were quite willing to discuss problems and offer advice, and future versions of the

VADS should take care of most of these current growing pains. Nevertheless, it was surprising to find them in a validated compiler, considering the extent of the DOD's compiler validation process.

Evaluation of Ruegg's Design

Ruegg describes the evolution of his design for AFIT_GKS quite thoroughly in his thesis. Although the Harris binding hints at implementing GKS according to its different levels, Ruegg and his thesis adviser chose to implement AFIT_GKS according to function. The primary reason for this is the extensive text facilities GKS requires even at the lowest GKS output level, "m". For an initial implementation of GKS, this method seems to be quite satisfactory. To realize all the text facilities would require a great deal of programming dealing solely with character drawing techniques, a task well removed from the overall objective of developing a working graphics system.

Although AFIT_GKS is a very large, unwieldy software system, it is made so by the size of the GKS standard itself. Some of Ruegg's design features, notably the internal variables, were handled as they were solely because of problems he encountered with the Rolm Data General's Ada compiler and, as it turns out, poorly designed portions of the Harris binding (as evidenced by the problems with the type MAGNITUDE discussed above). That Ruegg was able to get AFIT_GKS to work at all given these starting conditions is quite amazing.

Ruegg's documentation did leave something to be desired. Although an overall file structure and dependency picture would not be required by a casual user of AFIT_GKS, it is essential to a maintainer of AFIT_GKS. This information was obtained only after considerable "software sleuthing." Ruegg's in-line documentation, on the other hand, was quite good; in fact, this enabled most of the "sleuthing" to be accomplished. The only complaint here was that about a third of each procedure's in-line documentation is not useful: for example, it was unnecessary to state that the code was written in Ada.

In summary, given the compiler and binding environment he faced, Ruegg's design for AFIT_GKS was quite sensible and it was good enough to retain throughout the course of this project.

IV. Problem Identification

With the failure of AFIT_GKS to be transported directly to AFIT's ASC computer, a new approach was needed to retain the original form and structure of Ruegg's design. The alternative of discarding all of Ruegg's code and of rewriting AFIT_GKS was unacceptable. At the same time, it was considered important to try to revise AFIT_GKS to conform to Harris's latest binding, Revision 3.

The decision to move to Revision 3 also carried some hope. Although most of the GKS data types defined in the new binding were essentially the same, there were substantial changes to the major utility packages. Since telephone conversations with Harris had indicated that they also had had problems with the older binding (Cuthbert, 1985; Furtney, 1985), it seemed likely that the compiler was having problems with those much-used utility packages. Perhaps making the changes to the utility packages while leaving the rest of AFIT_GKS the same would contribute to a successful execution.

The first step in the new approach, however, was to attempt to isolate the specific source code that caused the "Illegal instruction" error, to see if that could help pinpoint which utility package was to blame.

Problem Isolation

By using the VADS' debugger utility, it was possible to discover that the "Illegal instruction" occurred in procedure

init_ws_state, located in file f27int_contrl_pb.a. Specifically, it occurred on the line where the initial transformation list is set to the default (identity) transformation:

```
e_norm_transformations.add_item(temp_trans,current_trans);
```

Using this line as the only line of a small test program, the files required to define all the data structures used in this line were assembled and stripped of everything that was not needed, so as to have the smallest amount of code possible that would still produce the error. The files all retained their original package structure; that is, separate compilation units remained separate. The resulting files are shown in Appendix F: six separate utility and type specification packages, and a seventh file containing the short test program.

Upon execution of the short test program, the UNIX error message "Bus error (core dumped)" was received. Even though the name of the error had changed (from "Illegal instruction"), it was felt that these results were unpredictable and the error was still being caused by the same problem.

In an attempt to see if the Verdix Product Support Office could solve the problem, a report containing these files and a description of the problem was prepared using the VADS "a.report" utility. This report was sent to Verdix using ARPANET, and although Verdix agreed that it was a most unusual and unexpected problem, the company provided little hope for any kind of "quick fix."

In the process of isolating this code, however, it became fairly apparent that the problem lay with the GKS_LIST_UTILITIES package, since the error is indeed caused during a call to a GKS_LIST_UTILITIES function, ADD_ITEM. Since the differences between the old (Revision 1) and new (Revision 3) versions of GKS_LIST_UTILITIES were quite substantial, it appeared promising to rewrite the GKS_LIST_UTILITIES package in accordance with Revision 3 specifications and rerun the test of the condensed code (Appendix F).

New GKS_LIST_UTILITIES Package

There were several major differences between the Revision 1 and Revision 3 GKS_LIST_UTILITIES specifications. The primary difference was that in Revision 1, the structure of the type LIST_OF was explicitly defined (as a variable length array within a record), whereas as in Revision 3, LIST_OF is declared a private type, leaving its definition up to the GKS programmer. The Harris binding suggests several implementations: a linked list, an access to a record with a varying size component, or a packed array of BOOLEAN.

Because of the varying size of the lists created by GKS_LIST_UTILITIES and, more importantly, due to the poor record of performance obtained so far with the variable-sized array method, a linked list structure was chosen for the implementation of the new GKS_LIST_UTILITIES. The private type LIST_OF was defined to be a record containing two components: an access type called HEAD which pointed to the

beginning of the linked list, and a NATURAL type called SIZE which would always contain the current number of elements in the list. The actual elements of the list, called NODEs, were also defined as a record containing two components: an OBJECT of the generic ELEMENT_TYPE, and an access type called NEXT which pointed to the next NODE in the list.

Some assumptions had to be made concerning the intended operation of the GKS_LIST_UTILITIES package that were not addressed by the Harris binding. For example, if an element to be added to a list were already in the list, should the element be put in a second time, or should an error message be returned somehow stating that the element was already in the list? And if an identical element was in a list several times and the DELETE_FROM_LIST function was called, which occurrence of the element should be deleted? Should an element be added to the head of the list, or the end of the list? Because of the lack of guidance from Harris, and because there were no known instances where an identical element would need to be placed in a list, the GKS_LIST_UTILITIES package was written to accept identical elements, and return the first occurrence of an identical element when the LIST_ELEMENT function was called. Also, all elements added would be placed on the head of the list for the sake of simplicity, and the function LIST_ELEMENT could be called in an application-dependent way to access the LIST in whatever order was required by a particular application.

The completed Revision 3 GKS_LIST_UTILITIES package is

shown in its entirety in Appendix J.

Results Using New GKS_LIST_UTILITIES Package

The completed GKS_LIST_UTILITIES package compiled successfully, but it needed validation testing. The first such test was to make it available to the condensed code (listed in Appendix F) that had caused the troublesome "Bus error" described earlier. All the files were recompiled, and this time upon execution the short program terminated normally with no Ada or UNIX error messages. This was certainly encouraging, and the first sign of success received in quite a long time, but it was hardly a complete test of the GKS_-LIST_UTILITIES package. A complete test program was written to test every function and procedure contained in GKS_LIST_-UTILITIES, and to print out intermediate results to insure that the test list was being handled correctly. This test was also successful.

With this new GKS_LIST_UTILITIES, it now seemed possible to go back and start the compiling process of AFIT_GKS once again. There would have to be some changes made to accommodate the revised procedure and function names (e.g., ADD_TO_-LIST vs. ADD_ITEM), but these changes were merely cosmetic in nature and could be handled quite easily.

V. A Successful Subset

As long as the entire list of AFIT_GKS files was eventually now going to be compiled once again, it seemed prudent to incrementally convert to the updated and more desirable Revision 3 binding. Since this conversion involved at most a rearrangement of the GKS_CONFIGURATION package and a few (mostly editorial) changes in the EXTERNAL_TYPES package, only a slight rearrangement of just the first few of Ruegg's files, and not of his entire file structure or source code, would be necessary.

Complete Switch to New Binding

As can be seen from Appendix B, files f02* to f07* are those files required to define all the GKS types and utility and configuration packages. The files following these are the actual implementation of AFIT_GKS, all of which use the structures and constants provided by f02* to f07*. It was desirable to retain the same numbering system during the switch to the new binding to avoid having to change all the file names.

Some changes to the file structure were nevertheless necessary. Since the Revision 3 binding split out the matrix functions of the old GKS_LIST_UTILITIES into a separate GKS_MATRIX_UTILITIES package, an extra file was necessary for this matrix package. Also, Harris had indicated (Furtney, 1985b) that the GKS_CONFIGURATION package should be the first

package to be compiled. But since a few of the GKS types needed to be defined and available prior to compilation of the GKS_CONFIGURATION package, still another package had to be added. The files were rearranged as shown below to handle the new binding but still retain the file structure of Ruegg's AFIT_GKS.

Ruegg's Files	Revision 3 Files
no equivalent	k00basic_types.a
no equivalent	k01gks_config_ps.a
f02gks_coord_psb.a	k02gks_coord_ps.a
f03gks_list_util_psb.a	k03gks_list_util_psb.a
f04gks_config_ps.a	k04gks_matrix_util_ps.a
f05ext_types_ps.a	k05ext_types_ps.a
f06int_types_ps.a	k06int_types_ps.a
f07int_vars_ps.a	k07int_vars_ps.a

In this way, all of the Revision 3 changes could be accommodated yet retain Ruegg's original file structure. Upgrading the remaining files to use the new GKS_LIST_UTILITIES package and the rest of the Revision 3 specifications could now begin.

Interfacing New Binding to Ruegg's Code

As expected, the editorial changes to the GKS types defined in the Revision 3 binding did not cause any great problems. What did come as a surprise was the extent of the changes required to handle the new GKS_LIST_UTILITIES package. These problems were caused simply because the types defined in the original Revision 1 GKS_LIST_UTILITIES package were not private. This open access allowed Ruegg to use those types in whatever manner he found convenient in order

to accomplish a particular routine. Although some of the changes required to access the new private data structures were quite simple to effect, others required considerable rewriting to accomplish. Consider these examples from file f12*:

1. In procedure d_ws_polyline_1, one of Ruegg's original lines of code reads as follows:

```
for i in 1..line_points.length loop
```

This line is easily converted for use by the Revision 3 GKS_LIST_UTILITIES by writing:

```
for i in 1..size_of_list( line_points) loop
```

2. Similarly, Ruegg's original line:

```
point1 := line_points.list( lower_index);
```

can be rewritten as:

```
point1 := list_element( lower_index, line_points);
```

3. But compare now what happens when that element of the list is found on the left side of an assignment statement. It is now not quite so easily converted. In procedure d_ws_polymarker_1, Ruegg's line:

```
marker_points.list(i) := marker_points.list(i) - dsp;
```

cannot be written for the new GKS_LIST_UTILITIES as:

```
list_element(i,marker_points) :=
                        list_element(i,marker_points) - dsp;
```

because "list_element" on the left-hand side does not point to an object; it is a function call that returns one. In this case, some significant rewriting has to be done. First, some temporary local variables have to be declared:

```
temp_point : POINT;
temp_list  : LIST_OF := NULL_LIST;
```

Now the original line can be rewritten as:

```
temp_point := list_element(i,marker_points) - dsp;
```

Then, the temporary point is added to the temporary list:

```
add_to_list( temp_point, temp_list);
```

Finally, the temp_list's value is given back to the original list:

```
marker_points := temp_list;
```

Thus, one line has expanded to five lines.

Because such changes were necessitated hundreds of times in the AFIT_GKS files, it quickly became apparent that it would not be possible to complete a total conversion of AFIT_GKS to use Revision 3's GKS_LIST_UTILITIES in the time available. Instead of attempting, as before, to go for an "all or none" conversion, working from the top of the AFIT_GKS file list down to the DEMO program, a plan to work

incrementally and iteratively from the DEMO program towards the top of the list was formed. In this way, progress on the conversion of AFIT_GKS to Revision 3 could be checked during the process of conversion, and a successful subset of AFIT_-GKS capabilities would always be available.

Testing the GKS Types and Ruegg's Types

Before beginning the conversion process, it seemed wise to verify that the packages containing all the GKS types, constants, and utilities, did in fact work as they were intended. This also seemed to be the best time to check for the proper functioning of all the types Ruegg designed for his implementation: the "internal_types" defined in file f06int_types.a.

Experience thus far with AFIT_GKS and the VADS had shown that quite often a section of code could be compiled and produce no errors, yet fail upon execution. For this reason, it seemed best to design a test program that would not only test the compilation of a particular type, but would also declare an object of each of those types. Many times before an object could be declared and pass through the compiler with no problem, but upon execution and the subsequent elaboration of that object an error would occur.

The test program thus produced declared objects of 107 different GKS types, all of which were defined in package EXTERNAL_TYPES. The program also declared objects of each of the 52 "internal_types" defined by Ruegg, and tested each of

the 55 instantiations of the generic GKS list, matrix, and coordinate_system utility packages. The test program was compiled without error, but upon execution immediately failed. By using the VADS source debugger extensively, the sources of error could then be located and corrected, one at a time.

The first problem encountered was a "numeric_error" which the debugger identified as originating with the type VARIABLE_CONNECTION_ID contained in package EXTERNAL_TYPES:

```
type VARIABLE_CONNECTION_ID( LENGTH: NATURAL := 0) is
    record
        CONNECT: CONNECTION_ID( 1..LENGTH);
    end record;
```

Objects of this type, according to the Harris binding, were to be declared without a discriminant value, letting the value for LENGTH take on the default value of 0. In this way, the LENGTH could then be dynamically altered during program execution. This is legal Ada code, and the VADS could compile it but would not execute it.

Discussions with the GKS binding team at Harris (Furtney, 1985a) indicated that they had also experienced problems with this type. They had received similar reports of problems from several other (unnamed) institutions, and it appeared that some Ada compilers could handle this type while others could not. Apparently the VADS compiler, in elaborating an object of this type, would try to set aside the memory space required for the largest possible instance of this

object. In this case, because NATURAL is a predefined subtype with a maximum value of INTEGER'LAST, VADS tries to build an array with a size of 1..INTEGER'LAST. This quickly uses up all available memory space. This problem was discussed by Barnes long before there were any serious Ada compilers (Barnes, 1982:268), and it should be handled more gracefully by the VADS. Harris avoided this problem by defining a type with a much more limited size to replace NATURAL, and they indicated that this change would probably appear in the next version of their binding.

A similar fix was made to AFIT_GKS. There are five other GKS types that are of essentially the same form as VARIABLE_CONNECTION_ID, and rather than declare a separate smaller type for each of them, one subtype was defined:

```
subtype SUB_NATURAL is NATURAL range 0..50;
```

The logical place for this definition was in the GKS_CONFIGURATION package, and can be changed by the user if he requires a range larger than 0..50. This new subtype was then substituted for NATURAL in the following types appearing in package EXTERNAL_TYPES:

```
VARIABLE_CONNECTION_ID
INPUT_STRING
TRANSFORMATION_PRIORITY_LIST
CHOICE_PROMPT_STRING
CHOICE_PROMPT_STRING_LIST
VARIABLE_SUBPROGRAM_NAME
```

Following these changes, the files were again recompiled

followed by the successful compilation of the test program. This time, when the test program was executed, a "numeric_error" was again produced, and the debugger pointed to the following type in package GKS_COORDINATE_SYSTEM:

```
type POINT_LIST( LENGTH: NATURAL:= 0) is
     record
        POINTS: POINT_ARRAY( 1..LENGTH);
     end record;
```

This was obviously the same type of problem encountered previously, and could have been handled quickly by replacing NATURAL with the smaller type SUB_NATURAL. However, objects of this type could conceivably be expected to grow to lengths much larger than the 50 allowed by SUB_NATURAL, so this type was handled separately. A new constant was declared, MAX_LENGTH_FOR_POINT_LIST, with a value of 1000, and placed, once again, in the GKS_CONFIGURATION package where it could be adjusted as required by the user. Then, in the generic GKS_COORDINATE_SYSTEM package, a new subtype was declared:

```
subtype INDEX is NATURAL
        range 0..MAX_LENGTH_FOR_POINT_LIST;
```

and the POINT_LIST type was rewritten as:

```
type POINT_LIST( LENGTH: INDEX:= 0) is
     record
        POINTS: POINT_ARRAY( 1..LENGTH);
     end record;
```

The next problem discovered by the type testing program was caused by the type VARIABLE_MATRIX_OF in the generic

```
package GKS_MATRIX_UTILITIES:

     type VARIABLE_MATRIX_OF( DX: NATURAL:= 0;
                              DY: NATURAL:= 0) is
          record
             MATRIX: MATRIX_OF( 1..DX, 1..DY);
          end record;
```

This again was the same type of problem, and was handled the same way as was POINT_LIST, using a new constant MAX_SIZE_-FOR_MATRIX in the GKS_CONFIGURATION package, a new subtype INDEX in GKS_MATRIX_UTILITIES with a range of 0..MAX_SIZE_-FOR_MATRIX, and replacing NATURAL with INDEX in VARIABLE_MA-TRIX_OF's definition.

Since this type was to be used not only for simple 2 x 3 transformation matrices but also for descriptions of complex hatch patterns, MAX_SIZE_FOR_MATRIX was initially set to a value of 500. However, upon compilation and execution of the files and test program, the UNIX error message "Segmentation fault (core dumped)" was produced, and the VADS debugger was of no help in pinpointing the cause. However, in realizing that a 500 x 500 matrix contains 250,000 elements, the much smaller value of 10 was tried for MAX_SIZE_FOR_MATRIX. This time the GKS_MATRIX_UTILITIES package worked. The "Segmenta-tion fault" appears to be another VADS problem that needs to be corrected. If 250,000 elements are too many for the VAX to handle, the VADS should handle the error, not UNIX.

The type testing program now worked all the way through, and the GKS types and packages could now be used with confi-dence. There remained one file to check before beginning

work on the GKS routines: f07*, which defined the variables use by Ruegg in his implementation. The only problem discovered here was also located with the debugger. This was also a storage problem, caused by the new Revision 3 binding.

In the old binding, there was a GKS_CONFIGURATION constant MAX_WS_ID, which Ruegg had given a value of 3. When he declared his objects:

```
U_WSS, U_DWS : array ( WS_ID) of DES_PTR;
```

this did not cause an error, because WS_ID had been defined as a range of 1..MAX_WS_ID. In the Revision 3 binding, however, there was no constant MAX_WS_ID defined, and WS_ID was defined simply as "new POSITIVE". So now this definition for U_WSS and U_DWS tried to set aside space for an array with a length of POSITIVE which quickly used all available memory space. To fix this problem, yet still retain Ruegg's original variable design, a constant MAX_WS_ID was added to the GKS_CONFIGURATION package, and the line above was rewritten as:

```
U_WSS, U_DWS : array ( 1..MAX_WS_ID) of DES_PTR;
```

During the testing process described above, there was one more instance of the troublesome VADS internal compiler error. It occurred during the compiling of file f05*, the EXTERNAL_TYPES package. Quite by accident, it was found that this internal error could be handled simply by recompiling the file without making a single change to it! This does not

sound at all reasonable, but was demonstrated several times.

Now all of the GKS types and packages, and Ruegg's types and global variables were working properly, and work could begin on the GKS routines themselves. The completed packages defining all the GKS constants, the GKS utility packages, and the GKS types, are shown in Appendices G through L.

GKS Routines Made Operable

The most suitable point to begin the conversion process was the "control" package, because it contains the functions first used by a GKS application program.

The first procedure to be called by an application program is OPEN_GKS(ERROR_FILE, AMOUNT_OF_MEMORY), where ERROR_FILE is a string containing the desired name for the error file (defaulting to the implementation-defined error file name), and AMOUNT_OF_MEMORY is the amount of available memory space. Similarly, the last GKS procedure to be called by an application program is CLOSE_GKS.

In this case, the program f46DEMO.a calls OPEN_GKS and CLOSE_GKS. The OPEN_GKS and CLOSE_GKS procedures are contained in f28contrl_psb.a, and OPEN_GKS in turn calls procedures located in f26int_contrl_ps.a and f27int_contrl_pb.a. CLOSE_GKS is completely "self-contained," and therefore makes no calls to any other procedures. These relationships are illustrated below:

f46*		f28*		f26*/f27*
DEMO	calls	OPEN_GKS	calls	init_open

calls CLOSE_GKS

OPEN_GKS in fact calls many other procedures in addition to those shown above, including the error-checking routines, and the initialization routines for the Tektronix 4014 and the Workstation Independent Segment Storage (WISS) device. But, in the attempt to convert iteratively larger segments of code, the only device addressed in this project was the Tektronix 4027A, and code for the other devices was commented out. Code for the error-checking routines was also commented out.

The first portion of OPEN_GKS to be converted and tested was the naming and opening of the external error file, which was handled in procedure OPEN_GKS itself. For some reason, Ruegg had written this code to name the file "gks_error" no matter what string was input in the call to OPEN_GKS, and it is not clear why he chose to do this. This was easily corrected, and several lines of text output were added to see if they would, in fact, appear in the file. Upon testing, an external file was created, and the text was output to it. At the same time the global variable "state_value" is set to "GKOP" to show that GKS is now open.

It should be noted here that the parameter AMOUNT_OF_-MEMORY is ignored in AFIT_GKS and so was not experimented with further. The Revision 1 binding stated that it could be "ignored in an environment supporting only static memory

management" (X3H3/83-95r1, 1984:67), and this statement is not clarified further in any of the later binding revisions.

The next procedure to be implemented was init_ws_state, which initializes the global variables defining the current workstation type, the current "pick" identification, and the current list of transformations. This procedure made two calls to instantiations of the GKS_LIST_UTILITIES package, and these lines were rewritten to reflect the changes to that utility package. No problems occurred with these changes.

The procedure init_dws_4027 was a very large and important one: it is the procedure which actually sets up the workstation description table for the Tektronix 4027A, defining the line types, marker types, colors and hatch pattern types, and display surface size available on that device. Quite a few changes were required here to update it to the Revision 3 binding. First, the references to all rectangle sizes had to be changed to reflect the Revision 3 changes in the GKS_COORDINATE_SYSTEM package. This occurred in six places. There were also references to 22 instantiations of the GKS_LIST_UTILITIES package, all of which needed to be converted to reflect the Revision 3 changes to that utility package. The lists created were those dealing with the colors, line types, marker types, and other attributes of the device.

One other problem arose when four lines referenced constants that used to be contained in the Revision 1 GKS_CONFIGURATION package. Although this problem could be remedied

by defining them here in procedure init_dws_4027, it would make more sense in the future to put them back into the GKS_CONFIGURATION package.

In all, 133 separate attributes of the Tektronix 4027A were defined and successfully stored in memory by the converted procedure init_dws_4027. This was quite a severe test of the GKS types, Ruegg's types, and the GKS utility packages, and conversion of the rest of AFIT_GKS certainly seems possible at this time.

The final procedure converted was the last procedure that would be called by a GKS application program: CLOSE_GKS. This procedure nulls out all the workstation state lists, sets the global variable "state_value" back to "GKCL" to show that GKS is now closed, and closes out the external error file. Conversion of this procedure was quite straightforward and worked successfully.

VI. Conclusion

Summary of Results

Although problems with the porting of AFIT_GKS to the AFIT ASC prevented the completion of the interface and evaluation portions of this project, many valuable insights into Ada, the Verdix VADS compiler and the Harris binding were gained. These are lessons that will not have to be relearned during any subsequent work on AFIT_GKS.

For example, the storage space allocation problems arising during execution have been solved as documented in Chapter V. Whether this is a problem with the VADS compiler or the Harris binding is not clear. Considering the problems reported by other institutions to Harris concerning storage allocation and the discussion of this subject by Barnes (Barnes, 1982:268), one is led to conclude that it is a binding problem. Other than in the GKS_CONFIGURATION package, however, the binding should not have to be modified depending on which particular machine it is operating.

The VADS is not completely faultless, nevertheless. The VADS error messages received by a programmer would not lead him to believe that there was a storage allocation problem. The many internal compiler errors discovered and the sometimes nonsensical means required to eliminate them point out some serious problems for the product support personnel at Verdix.

Based on the problems experienced in this project, it

can be said that Ada has not yet reached the degree of portability envisioned by its designers. Both the Rolm Data General's Ada and the Verdix VADS are DOD-validated Ada compilers, yet Ruegg's code had to undergo significant modification before it was found acceptable by the VADS. These problems and what solutions were discovered are documented in this project.

Probably the most important discovery is the fact that there are apparently different interpretations of the rules for the Ada language as laid down in the Language Reference Manual. This was illustrated by problems with some of the procedures Ruegg had written; for example, the procedures in which Ruegg had momentarily lapsed into Pascal instead of Ada and declared some superfluous local variables which were then caught by the VADS but apparently not by the Rolm Data General's Ada. Another example was the procedure where Ruegg had incorrectly used an OUT parameter: the Rolm Data General's Ada apparently accepted it while the VADS objected to it. One would expect the same code to be handled in the same way by two DOD-validated compilers.

The Verdix VADS compiler was overall quite easy to use. There are obviously still some "bugs" in it, as evidenced by the internal compiler errors encountered, but the communication lines to the product support technicians at Verdix were open and well-exercised during the course of this project. The existing library and debugging facilities of the VADS, and the planned enhancements to it, make it quite easy for

the VADS Ada programmer to develop software systems by freeing him from the drudgery of library maintenance and providing him the means to identify possible sources of error in his code.

As Ruegg has concluded (Ruegg, 1984: 5.1), Ada is indeed quite capable of handling computer graphics. Especially attractive to an implementation of GKS in Ada is Ada's unique generic package capability, preventing duplications of near-identical code to perform similar functions on items of different types. However, the problems with the actual Ada implementation of a standard as large as GKS are likely to continue until the different Ada compilers have matured to the point where "legal" Ada code will run on any compiler without modification.

Recommendations for Further Study

There is no reason to abandon AFIT_GKS at this point of its development on an AFIT computer system. As a result of this study, a firm foundation of GKS types, routines, and utility packages has been designed, implemented, thoroughly tested, and documented. Much information and documentation has been obtained, as well as very useful sources of further information (Appendix M). This foundation can serve as a baseline for continued conversion of AFIT_GKS, or as a ground floor for an entirely new implementation of GKS in Ada.

If one were to continue conversion of AFIT_GKS in its present form, the most logical place to begin would be to

continue to convert the code from the bottom up, starting in the "control" package. In retrospect, the initial attempts at conversion, starting at the very top of the AFIT_GKS file list and working towards an "all or none" conversion, was a very poor method to choose. More progress was made in a shorter amount of time by working from the "bottom up" and getting ever larger sections of code to work. The lines of code commented out in this process were clearly prefixed with "--*" to distinguish them from comments made by Ruegg, and all other changes made to Ruegg's code was also well annotated with in-line documentation.

Whether in continuing conversion of AFIT_GKS or in beginning a new implementation, consideration must be given to the fact that many of the various documents used in this project are still in an almost continuous state of change. Although GKS has become an official American National Standard, it has not yet been published in its final form, and it is not inconceivable that some changes will yet be made to it. Similarly, at this writing, the Harris binding is about to enter its fourth revision, and it is entirely possible that some changes will be made to handle the storage problems discussed earlier.

Looking even further ahead, interfacing of the Raster Technologies Model One would be an invaluable addition to an implementation of GKS in Ada. Especially attractive is the Model One's ability to perform clipping routines in onboard firmware. The terminals currently addressed by AFIT_GKS do

this in large amounts of code on the host machine, taking up considerable memory space and computing time. Because of its tremendous on-board capabilities, the Model One should in fact be easier to interface than the older display terminals used by AFIT_GKS. Any work in this area should be done in accordance with the Computer Graphics Interface standard currently being developed by the American National Standards Institute.

Finally, any implementation of GKS in Ada must be subjected to some form of testing and evaluation. The test tape provided by the National Bureau of Standards is only a first step towards a standard way of testing GKS implementations, and it will begin to take a more definite form in the months and years ahead. Any implementation work done in the future should be accomplished with eventual certification by the NBS as a goal; therefore, the programmer must keep an eye on the development of the Bureau's certification process.

As a minimum, an implementation should be able to display the minimum line types, marker types, and interior styles mandated by GKS. The first few tests of the initial test program accomplish this. The available text styles are also tested by the test programs, but AFIT_GKS at this time has only a limited text capability. Successful and meaningful testing of AFIT_GKS's text capabilities could only occur after a considerable amount of new code was written to supply those capabilities.

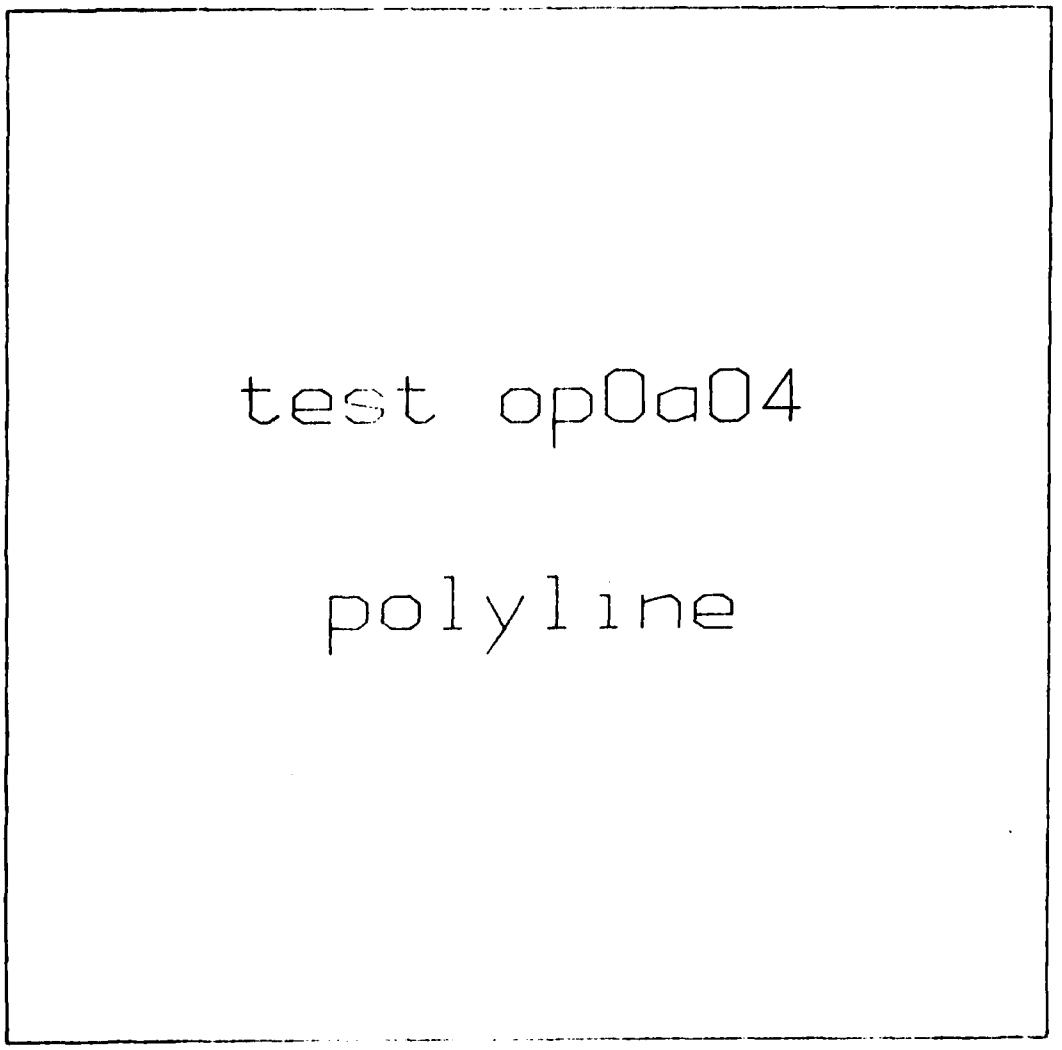
Computer graphics is here to stay, and it will continue

to play an ever-increasing role in Air Force information and weapon systems and educational institutions. As the Air Force and the other military services move towards across-the-board use of the Ada programming language, the need for Ada-based graphics capabilities will grow. Now is the time to develop those capabilities, and AFIT_GKS can become a useful vehicle for that development.

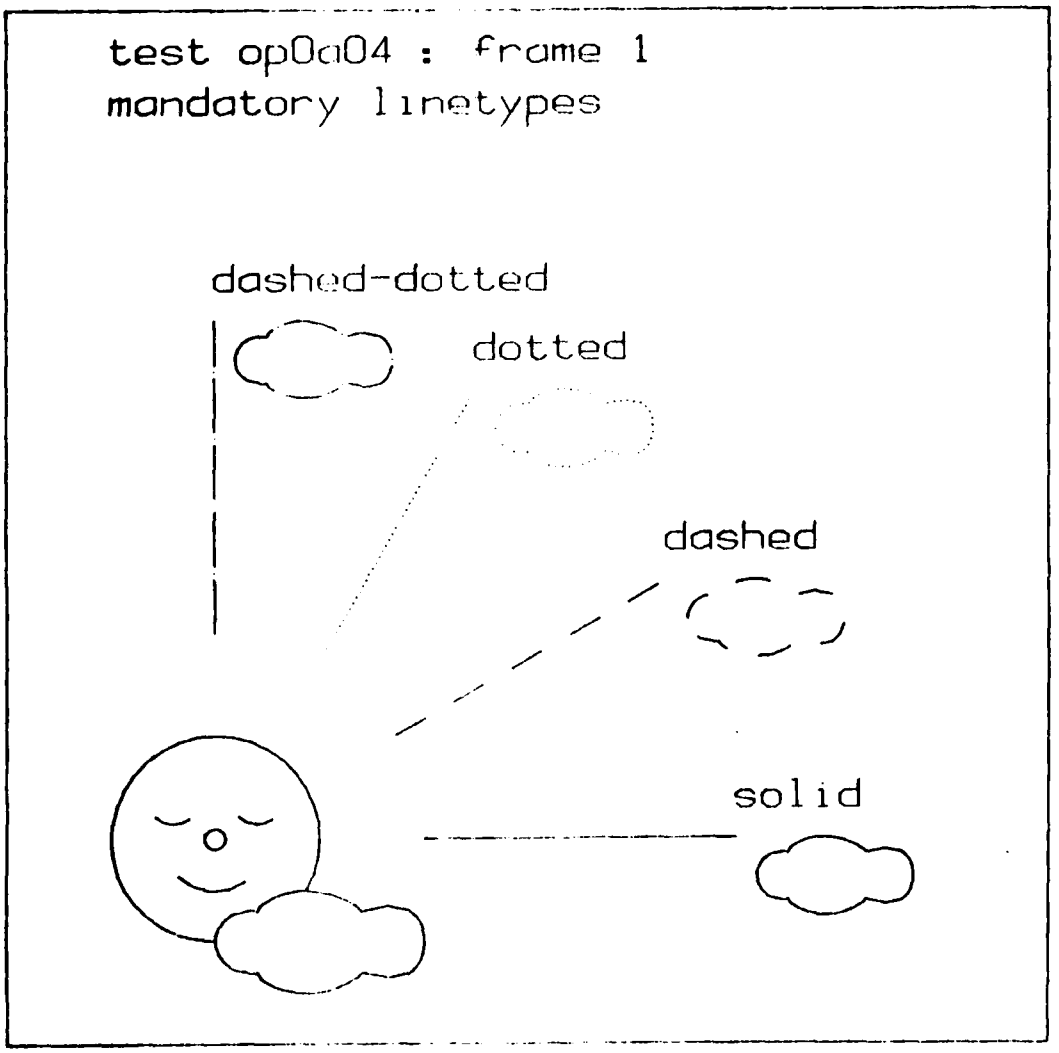
Appendix A:

Example Certification Output

Source: (National Bureau of Standards, 1985)



Test op0a04: Title frame



Test op0a04: Frame 1: Linetypes

Appendix B:

Tape Transfer Utilities

NOTE: These tape transfer utilities were written by Mr. W. Nicholas Miller, AFIT Computer Technical Representative, to handle the the transfer of the AFIT_GKS files from the Rolm Data General tape to the Academic Support Computer (ASC).

TO TRANSFER FROM THE TAPE TO THE SSC:

This UNIX script takes files from the tape and places them on disk with a filename of "file.x" where x is the file's position in the sequence of files.

```
#
@ N=1
while ($n < 47)
echo file.$n
dd if=/dev/nrmt8 of=file.$n ibs=1024 obs=80 cbs=80 conv=block
@ n++
end
```

TO CONVERT FILES ON ASC AFTER TRANSFER WITH UUCP:

This UNIX script takes the files and inserts linefeeds at the proper places and renames the converted files "filex" where x is the file's position in the sequence of files.

```
#
@ n=1
while ($n < 47)
echo file.$n
dd if=file.$n of=file$n obs=80 cbs=80 conv=unblock
@ n++
end
```

Appendix C:

AFIT_GKS File and Unit Names

The suffix to each of the file names (ps, pb, or pbs) identifies whether a file contains a package specification, a package body or both. This naming convention allows easy identification of the contents of a file, yet allows the files to be handled easily in UNIX commands using the '*' wild card character instead of typing the entire file name (e.g., f11*).

UNIX filename	Ada unit name
f01ruegg_code	(list of files)
f02gks_coord_pbs.a	GKS_COORDINATE_SYSTEM
f03gks_list_util_psb.a	GKS_LIST_UTILITIES
f04gks_config_ps.a	GKS_CONFIGURATION
f05ext_types_ps.a	EXTERNAL_TYPES
f06int_types_ps.a	INTERNAL_TYPES
f07int_vars_ps.a	INTERNAL_VARS
f08drive_1_psb.a	DRIVE_1
f09int_ws1_ps.a	int_ws_1
f10int_ws1_pb.a	int_ws_1
f11ws_1_ps.a	ws_1
f12ws_1_pb.a	ws_1
f13drive_2_ps.a	DRIVE_2
f14drive_2_pb.a	DRIVE_2
f15int_ws2_ps.a	int_ws_2
f16int_ws2_pb.a	int_ws_2
f17ws_2_ps.a	ws_2
f18ws_2_pb.a	ws_2
f19int_ws3_ps.a	int_ws_3
f20int_ws3_pb.a	int_ws_3
f21ws_3_ps.a	ws_3
f22ws_3_pb.a	ws_3
f23error_ps.a	error
f24error_pb.a	error
f25err_handl_psb.a	error_handling
f26int_contrl_ps.a	int_control
f27int_contrl_pb.a	int_control
f28control_psb.a	control
f29prim_ps.a	primitives
f30prim_pb.a	primitives

f31set_prim_psb.a	set_prim
f32represent_psb.a	represent
f33transform_psb.a	transform
f34segments_ps.a	segments
f35segments_pb.a	segments
f36set_input_ps.a	set_input
f37set_input_pb.a	set_input
f38input_psb.a	input
f39set_trans_psb.a	set_transform
f40emergency_psb.a	emergency
f41inq_attr_psb.a	inq_attributes
f42inq_represent_psb.a	inq_represent
f43inq_facil_psb.a	inq_facilities
f44inq_segment_psb.a	inq_segment
f45AFIT_GKS.a	AFIT_GKS (procedure)
f46DEMO.a	DEMO (procedure)

Appendix D:

AFIT_GKS File Dependencies

This chart shows what files need to be in the Ada library prior to compilation of any particular file. Any files below and to the right of any particular file must be compiled first. Seven levels are shown: -, --, ---, *, **, ***, and +. The suffixes "ps" and "pb" indicate "package specification" and "package body," respectively.

```
- DEMO

    -- external_types
         --- gks_list_utilities
         --- gks_coordinate_system
         --- gks_configuration
              * core_functions

    -- control
         --- gks_configuration
         --- external_types
         --- internal_types
              * external_types
              * gks_list_utilities
              * gks_coordinate_system
              * gks_configuration
         --- internal_vars
              * gks_configuration
              * external_types
              * internal_types
              * tty_io
         --- ws_1
              * gks_configuration
              * external_types
              * internal_types
              * internal_vars
              * int_ws_1
                   ** int_ws_ps
                        *** external_types
                        *** internal_types
                        *** internal_vars
                   ** int_ws_pb
                        *** gks_list_utilities
                        *** external_types
                        *** internal_types
                        *** internal_vars
                        *** drive_1
                        *** numeric_io
                        *** tty_io
```

```
        * gks_list_utilities
        * drive_1
            ** gks_coordinate_system
            ** gks_list_utilities
            ** external_types
            ** internal_types
            ** internal_vars
            ** tty_io
            ** numeric_io
        * numeric_io
        * tty_io
    --- ws_2
        * ws_2_ps
            ** gks_configuration
            ** external_types
            ** internal_types
            ** internal_vars
            ** int_ws_2
                *** int_ws_2_ps
                    + external_types
                    + internal_types
                    + internal_vars
                *** int_ws_2_pb
                    + gks_list_utilities
                    + external_types
                    + internal_types
                    + internal_vars
                    + drive_2
                    + numeric_io
                    + tty_io
        * ws_2_pb
            ** gks_list_utilities
            ** external_types
            ** internal_types
            ** internal_vars
            ** drive_2
                *** drive_2_ps
                    + external_types
                    + internal_types
                *** drive_2_pb
                    + gks_coordinate_system
                    + gks_list_utilities
                    + external_types
                    + internal_types
                    + internal_vars
                    + tty_io
                    + numeric_io
            ** numeric_io
            ** int_ws_2
            ** tty_io

    --- ws_3
        * ws_3_ps
```

```
                ** gks_configuration
                ** external_types
                ** internal_types
                ** internal_vars
                ** int_ws_3
                     *** int_ws_3_ps
                          + external_types
                          + internal_types
                          + internal_vars
                     *** int_ws_3_pb
                          + gks_list_utilities
                          + external_types
                          + internal_types
                          + internal_vars
           * ws_3_pb
                ** gks_list_utilities
                ** external_types
                ** internal_types
                ** internal_vars
                ** ws_1
                ** ws_2
                ** int_ws_3
                ** tty_io
      --- error
           * error_ps
                ** external_types
                ** internal_types
                ** internal_vars
           * error_pb
                ** external_types
                ** internal_types
                ** internal_vars
      --- error_handling
           * external_types
           * internal_types
           * internal_vars
           * tty_io
      --- int_control
           * int_control_ps
                ** external_types
           * int_control_pb
                ** gks_configuration
                ** external_types
                ** internal_types
                ** internal_vars
                ** ws_1
                ** ws_2
                ** int_ws_1
                ** ws_3
                ** error
                ** error_handling
      --- tty_io
```

```
-- primitives
     --- prim_ps
          * external_types
     --- prim_pb
          * external_types
          * internal_types
          * internal_vars
          * ws_1
          * int_ws_1
          * ws_2
          * ws_3
          * control
          * error
          * error_handling

-- set_prim
     --- external_types
     --- internal_types
     --- internal_vars
     --- error
     --- error_handling

-- transform
     --- external_types
     --- internal_types
     --- internal_vars
     --- error
     --- error_handling
     --- control
```

Appendix E:

Excerpt: GKS_COORDINATE_SYSTEM

NOTE: The following excerpt is from Ruegg's original GKS_COORDINATE_SYSTEM (as defined by Harris binding Revision 1), with all comments therein made by Ruegg.

```
generic
 type COORDINATE is digits <>;
  -- Coordinates in the system are floating point value
  -- Values on both axes are of the same type.

package GKS_COORDINATE_SYSTEM is

  subtype MAGNITUDE is COORDINATE range
   COORDINATE'SMALL .. abs(COORDINATE'LAST -
                           COORDINATE'FIRST);
 -- Defines the length of an object in the coordinate
 -- system.  In GKS, all such values must be greater
 -- than zero.
 -- this is not following the new standard because the Rolm
 -- will not accept it!  Therefore I followed the old
 -- standard

-- the new standard is
--   MAGNITUDE_PRECISION : CONSTANT := 6;
--
--   type MAGNITUDE_BASE_TYPE is digits MAGNITUDE_PRECISION;
--
-- subtype MAGNITUDE is MAGNITUDE_PRECISION range
--   MAGNITUDE_BASE_TYPE( COORDINATE' SAFE_SMALL).
--   MAGNITUDE_BASE_TYPE(ABS (COORDINATE' LAST_
--                            COORDINATE' FIRST);
-- I think this is a bad type Magnitude
 .
 .
 .
end GKS_COORDINATE_SYSTEM;
```

After the "new standard" mentioned by Ruegg was uncommented and the typographical errors corrected, the lines dealing with type MAGNITUDE appeared like this:

```
subtype MAGNITUDE is MAGNITUDE_BASE_TYPE range
   MAGNITUDE_BASE_TYPE( COORDINATE' SAFE_SMALL)..
   MAGNITUDE_BASE_TYPE(ABS (COORDINATE' LAST -
                            COORDINATE' FIRST);
```

Appendix F:

Error-producing Source Code

NOTE: This is the extremely condensed source code that was determined to produce the "Illegal instruction" problem during porting of AFIT_GKS to the AFIT ASC computer using the VADS compiler.

```
----- file k02coord.a -----

     generic
         type COORDINATE is digits <>;

     package GKS_COORDINATE_SYSTEM is
         type LIMITS is
             record
                 MIN : COORDINATE;
                 MAX : COORDINATE;
             end record;
         type RECTANGLE is
             record
                 X : LIMITS;
                 Y : LIMITS;
             end record;
     end GKS_COORDINATE_SYSTEM;

----- file k03list.a -----

     generic
          type ITEM_TYPE is private;

     package GKS_LIST_UTILITIES is
          MAX_INDEX : constant := 50;
          type INDEX is range 0..MAX_INDEX;
          type ARRAY_OF is array (INDEX range <>) of
               ITEM_TYPE;
          type LIST_OF ( LENGTH : INDEX := 0) is
               record
                    LIST : ARRAY_OF( 1..LENGTH);
               end record;
          procedure ADD_ITEM(    ITEM : ITEM_TYPE;
                                 TO_LIST : in out LIST_OF);
     end GKS_LIST_UTILITIES;

     package body GKS_LIST_UTILITIES is
          procedure ADD_ITEM(    ITEM : ITEM_TYPE;
                                 TO_LIST : in out LIST_OF) is
          NEW_LIST : LIST_OF( TO_LIST.LENGTH + 1);
      -- (cont next page)
```

```
            begin
                 for i in TO_LIST.LIST'RANGE loop
                     NEW_LIST.LIST(i) := TO_LIST.LIST(i);
                 end loop;
                 NEW_LIST.LIST(TO_LIST.LIST'LAST+1) := ITEM;
                 TO_LIST := NEW_LIST;
            end ADD_ITEM;
       end GKS_LIST_UTILITIES;

-----  file k04config.a -----

       package GKS_CONFIGURATION is
            PRECISION : constant INTEGER := 6;
            MAX_TRANSFORMATION_NUMBER : constant INTEGER := 20;
       end GKS_CONFIGURATION;

-----  file k05ext_types.a -----

       with gks_list_utilities;
       with gks_coordinate_system;
       with gks_configuration;
       package EXTERNAL_TYPES is
            use gks_configuration;
            type WC_TYPE is digits PRECISION;
            package WC is new
                    GKS_COORDINATE_SYSTEM( WC_TYPE);
            type TRANSFORMATION_NUMBER is range
                 0..MAX_TRANSFORMATION_NUMBER;
            package TRANSFORMATION_NUMBERS is new
                    GKS_LIST_UTILITIES( TRANSFORMATION_NUMBER);
       end EXTERNAL_TYPES;

-----  file k06inttypes.a -----

       with external_types;
       with gks_list_utilities;
       with gks_coordinate_system;
       with gks_configuration;
       package INTERNAL_TYPES is
            use gks_configuration;
            use external_types;
            type e_norm_transformation is
              record
                PRIORITY : TRANSFORMATION_NUMBER := 0;
                TRANSFORMATION : TRANSFORMATION_NUMBER := 1;
                WINDOW : WC.RECTANGLE :=
                         (X => (MIN => 0.0, MAX => 1.0),
                          Y => (MIN => 0.0, MAX => 1.0));
              end record;
            package e_norm_transformations is new
                  GKS_LIST_UTILITIES( e_norm_transformation);
       end INTERNAL_TYPES;
```

```
----- k07intvars.a -----

     with gks_configuration;
     with external_types;
     with internal_types;
     package INTERNAL_VARS is
          use gks_configuration;
          use external_types;
          use internal_types;

       CURRENT_TRANS_NUM : TRANSFORMATION_NUMBER := 0;
       CURRENT_TRANS_INDEX: E_NORM_TRANSFORMATIONS.INDEX:=1;
       CURRENT_TRANS : E_NORM_TRANSFORMATIONS.LIST_OF;
     end INTERNAL_VARS;

----- file k08.a -----

     with internal_types;
     with internal_vars;

     procedure k08 is

          use internal_types;
          use internal_vars;

          temp_trans : e_norm_transformation;

        begin
          e_norm_transformations.add_item( temp_trans,
                                           current_trans);

     end k08;

-------------------------
```

Appendix G:

Package BASIC_TYPES

```
------------------------------------------------------------
--                  GKS Basic Types Package                --
--               Harris Binding r3, 18 Aug 85              --
--                                                         --
--                 Kevin E. Leinbach, GSO-85D              --
------------------------------------------------------------
-- This package provides only those GKS types which
-- are required to compile the GKS_CONFIGURATION
-- package, which should be the first package to be
-- compiled in any GKS implementation, according to
-- Ms. Barbara Furtney, Harris Corporation, 18 Aug 85.

package BASIC_TYPES is

     ------------------------------------------------------
     type LOCATOR_PROMPT_ECHO_TYPE is new INTEGER;

     -- Level mb

     -- Defines the locator prompt and echo types supported
     -- by the implementation.
     ------------------------------------------------------
     type STROKE_PROMPT_ECHO_TYPE is new INTEGER;

     -- Level mb

     -- Defines the stroke prompt and echo types
     ------------------------------------------------------
     type VALUATOR_PROMPT_ECHO_TYPE is new INTEGER;

     -- Level mb

     -- Defines the possible range of valuator prompt and
     -- echo types
     ------------------------------------------------------
     type CHOICE_PROMPT_ECHO_TYPE is new INTEGER;

     -- Level mb

     -- Defines the choice prompt and echo types
     ------------------------------------------------------
     type PICK_PROMPT_ECHO_TYPE is new INTEGER;

     -- Level mb

     -- Defines the pick prompt and echo types
     ------------------------------------------------------
```

```
        type STRING_PROMPT_ECHO_TYPE is new INTEGER;

        -- Level mb

        -- Defines the string prompt and echo types
        ------------------------------------------------------

end BASIC_TYPES;
```

Appendix H:

Package GKS_CONFIGURATION

```
------------------------------------------------------------
--                GKS Configuration Package                --
--              Harris Binding r3, 18 Aug 85               --
--                                                         --
--                Kevin E. Leinbach, GSO-85D               --
------------------------------------------------------------
with BASIC_TYPES; use BASIC_TYPES;

package GKS_CONFIGURATION is

     -- The following constants and types are not defined by
     -- the Harris GKS binding, but are required to avoid
     -- run-time errors using the Verdix Ada Compiler.

     MAX_LENGTH_FOR_POINT_LIST : constant := 1000;
     -- used by:
     --    GKS_COORDINATE_SYSTEM: type POINT_LIST

     MAX_SIZE_FOR_MATRIX : constant := 10;
     -- used by:
     --    GKS_MATRIX_UTILITIES: type VARIABLE_MATRIX_OF

     subtype SUB_NATURAL is NATURAL range 0..50;
     -- used by: EXTERNAL_TYPES:
     --            type VARIABLE_CONNECTION_ID
     --            type INPUT_STRING
     --            type VARIABLE_SUBPROGRAM_NAME
     --            type TRANSFORMATION_PRIORITY_LIST
     --            type CHOICE_PROMPT_STRING
     --            type CHOICE_PROMPT_STRING_LIST

     MAX_WS_ID : constant := 3;
     -- used by:
     --    (Ruegg's) INTERNAL_VARS:
     --                  U_WSS, U_DWS
     ------------------------------------------------------
     -- The following constants are all "implementation-
     -- defined".  Where an equivalent constant existed in
     -- Ruegg's original AFIT_GKS code, the same value is
     -- used here and marked with a '--*'.  For those
     -- constants that appear only in Revision 3 of the
     -- Harris binding, an arbitrary 'safe' value is
     -- selected.

     MAX_MEMORY_UNITS      --*
                   : constant
                   := 2;
```

```
PRECISION                --*
                : constant
                := 5;

DEFAULT_ERROR_FILE     --* (hardwired in Ruegg's code)
                : constant STRING
                := "gks_error";
MAX_WS_TYPE              --*
                : constant
                := 3;
MAX_NUMBER_OPEN_WS
                : constant
                := 1;
MAX_NUMBER_ACTIVE_WS
                  constant
                := 1;
MAX_NUMBER_SEGMENT_WS --*
                : constant
                := 50;
MAX_NUMBER_NORMALIZATION_TRANSFORMATION_NUMBER  --*
                : constant
                := 20;

-- The following type appears here to preclude the
-- error described in LRM 3.4(15).

type SCALE_FACTOR is digits PRECISION;

-- The following constants define the prompt and echo
-- types supported by GKS, and appear here as they
-- appear in Harris Binding r3.

DEFAULT_LOCATOR
                : constant LOCATOR_PROMPT_ECHO_TYPE
                := 1;
DEFAULT_STROKE
                : constant STROKE_PROMPT_ECHO_TYPE
                := 1;
DEFAULT_VALUATOR
                : constant VALUATOR_PROMPT_ECHO_TYPE
                := 1;
DEFAULT_CHOICE
                : constant CHOICE_PROMPT_ECHO_TYPE
                := 1;
DEFAULT_PICK
                : constant PICK_PROMPT_ECHO_TYPE
                := 1;
DEFAULT_STRING
                : constant STRING_PROMPT_ECHO_TYPE
                := 1;
```

```
        CROSS_HAIR_LOCATOR
                        : constant LOCATOR_PROMPT_ECHO_TYPE
                        := 2;
          TRACKING_CROSS_LOCATOR
                        : constant LOCATOR_PROMPT_ECHO_TYPE
                        := 3;
          RUBBER_BAND_LINE_LOCATOR
                        : constant LOCATOR_PROMPT_ECHO_TYPE
                        := 4;
          RECTANGLE_LOCATOR
                        : constant LOCATOR_PROMPT_ECHO_TYPE
                        := 5;
          DIGITAL_LOCATOR
                        : constant LOCATOR_PROMPT_ECHO_TYPE
                        := 6;

          DIGITAL_STROKE
                        : constant STROKE_PROMPT_ECHO_TYPE
                        := 2;
          MARKER_STROKE
                        : constant STROKE_PROMPT_ECHO_TYPE
                        := 3;
          LINE_STROKE
                        : constant STROKE_PROMPT_ECHO_TYPE
                        := 4;

          GRAPHICAL_VALUATOR
                        : constant VALUATOR_PROMPT_ECHO_TYPE
                        := 2;
          DIGITAL_VALUATOR
                        : constant VALUATOR_PROMPT_ECHO_TYPE
                        := 3;

          PROMPT_ECHO_CHOICE
                        : constant CHOICE_PROMPT_ECHO_TYPE
                        := 2;
          STRING_PROMPT_CHOICE
                        : constant CHOICE_PROMPT_ECHO_TYPE
                        := 3;
          STRING_INPUT_CHOICE
                        : constant CHOICE_PROMPT_ECHO_TYPE
                        := 4;
          SEGMENT_CHOICE
                        : constant CHOICE_PROMPT_ECHO_TYPE
                        := 5;

          GROUP_HIGHLIGHT_PICK
                        : constant PICK_PROMPT_ECHO_TYPE
                        := 2;
          SEGMENT_HIGHLIGHT_PICK
                        : constant PICK_PROMPT_ECHO_TYPE
                        := 3;
      end GKS_CONFIGURATION;
```

Appendix I:

Package GKS_COORDINATE_SYSTEM

```
-------------------------------------------------------------
--          GKS Generic Coordinate System Package          --
--              Harris Binding r3, 18 Aug 85               --
--                                                         --
--               Kevin E. Leinbach, GSO-85D                --
-------------------------------------------------------------
with GKS_CONFIGURATION; use GKS_CONFIGURATION;

generic
     type COORDINATE_COMPONENT_TYPE is digits <>;

package GKS_COORDINATE_SYSTEM is

     type POINT is
          record
               X: COORDINATE_COMPONENT_TYPE;
               Y: COORDINATE_COMPONENT_TYPE;
          end record;

     type POINT_ARRAY is array (POSITIVE range <> of POINT);

     -- This nect subtype is required to prevent run-time
     -- errors using VADS.  It is not part of the GKS
     -- binding by Harris.

     subtype INDEX is NATURAL range 0 ..
                         MAX_LENGTH_FOR_POINT_LIST;

     type POINT_LIST ( LENGTH : INDEX := 0) is
          record        -- INDEX replaces binding's NATURAL
               POINTS: POINT_ARRAY( 1 .. LENGTH);
          end record;

     type VECTOR is new POINT;

     type RECTANGLE_LIMITS is
          record
             XMIN: COORDINATE_COMPONENT_TYPE;
             XMAX: COORDINATE_COMPONENT_TYPE;
             YMIN: COORDINATE_COMPONENT_TYPE;
             YMAX: COORDINATE_COMPONENT_TYPE;
          end record;

     type MAGNITUDE_BASE_TYPE is digits PRECISION;
```

```
        subtype MAGNITUDE is MAGNITUDE_BASE_TYPE range
            COORDINATE_COMPONENT_TYPE'SAFE_SMALL
            .. COORDINATE_COMPONENT_TYPE'SAFE_LARGE;

        type SIZE is
             record
                XAXIS: MAGNITUDE;
                YAXIS: MAGNITUDE;
             end record;

        type RANGE_OF_MAGNITUDES is
             record
                MIN: MAGNITUDE;
                MAX: MAGNITUDE;
             end record;

end GKS_COORDINATE_SYSTEM;
```

Appendix J:

Package GKS_LIST_UTILITIES

```
-------------------------------------------------------------
--            GKS Generic List Utility Package             --
--              Harris Binding r3, 18 Aug 85               --
--                                                         --
--               Kevin E. Leinbach, GSO-85D                --
-------------------------------------------------------------
generic
    type ELEMENT_TYPE is private;

package GKS_LIST_UTILITIES is

    type LIST_OF is private;

    NULL_LIST : constant LIST_OF;

    procedure ADD_TO_LIST( ELEMENT : in  ELEMENT_TYPE;
                           LIST    : in out LIST_OF);

    procedure DELETE_FROM_LIST( ELEMENT : in ELEMENT_TYPE;
                                LIST    : in out LIST_OF);

    function SIZE_OF_LIST( LIST : in LIST_OF) return NATURAL;

    function IS_IN_LIST( ELEMENT : in ELEMENT_TYPE;
                         LIST    : in LIST_OF)
                                        return BOOLEAN;

    function LIST_ELEMENT( I    : in POSITIVE;
                           LIST : in LIST_OF)
                                    return ELEMENT_TYPE;

    type LIST_VALUES is array (POSITIVE range <>)
                                          of ELEMENT_TYPE;

    function LIST( VALUES : in LIST_VALUES) return LIST_OF;

    private

        type NODE;

        type PTR is access NODE;

        type NODE is
            record
                OBJECT : ELEMENT_TYPE;
                NEXT   : PTR;
            end record;
```

```
        type LIST_OF is
            record
                HEAD : PTR;
                SIZE : NATURAL;
            end record;

        NULL_LIST : constant LIST_OF := ( null, 0);

end GKS_LIST_UTILITIES;

package body GKS_LIST_UTILITIES is

    ------------------------------------------------------
    procedure ADD_TO_LIST( ELEMENT : in  ELEMENT_TYPE;
                           LIST    : in out LIST_OF) is
        P : PTR;

        begin

            P := new NODE;
            P.OBJECT := ELEMENT;
            P.NEXT   := LIST.HEAD;
            LIST.HEAD := P;
            LIST.SIZE := LIST.SIZE + 1;

        end ADD_TO_LIST;
    ------------------------------------------------------
    procedure DELETE_FROM_LIST( ELEMENT : in ELEMENT_TYPE;
                                LIST    : in out LIST_OF) is

        P, L : PTR := LIST.HEAD;

        begin

            while P /= null and then P.OBJECT /= ELEMENT loop
                L := P;
                P := P.NEXT;
            end loop;
            if P /= null
                then if P = LIST.HEAD
                        then LIST.HEAD := P.NEXT;
                             LIST.SIZE := LIST.SIZE - 1;
                        else L.NEXT := P.NEXT;
                             LIST.SIZE := LIST.SIZE - 1;
                     end if;
            end if;

    end DELETE_FROM_LIST;
    ------------------------------------------------------
    function SIZE_OF_LIST(LIST: in LIST_OF) return NATURAL is
        begin
            return LIST.SIZE;
        end SIZE_OF_LIST;
```

```
    ------------------------------------------------------------
    function IS_IN_LIST( ELEMENT : in ELEMENT_TYPE;
                         LIST    : in LIST_OF)
                                          return BOOLEAN is

        P : PTR := LIST.HEAD;
        it_is : BOOLEAN := false;

        begin

            while P /= null loop
                it_is := ( ELEMENT = P.OBJECT);
                if it_is
                    then exit;
                    else P := P.NEXT;
                end if;
            end loop;
            return it_is;

        end IS_IN_LIST;
    ------------------------------------------------------------
    function LIST_ELEMENT( I    : in POSITIVE;
                           LIST : in LIST_OF)
                                     return ELEMENT_TYPE is
        P : PTR := LIST.HEAD;

        begin

            for counter in 1..LIST.SIZE - 1 loop
                P := P.NEXT;
            end loop;
            return P.OBJECT;

    end LIST_ELEMENT;
    ------------------------------------------------------------
    function LIST( VALUES : in LIST_VALUES)
                                        return LIST_OF is
        NEW_LIST : LIST_OF := NULL_LIST;

        begin

            for i in VALUES'range loop
                ADD_TO_LIST( VALUES( i), NEW_LIST);
            end loop;
            return NEW_LIST;

        end LIST;
    ------------------------------------------------------------

end GKS_LIST_UTILITIES;
```

Appendix K:

Package GKS_MATRIX_UTILITIES

```
-------------------------------------------------------------
--            GKS Generic Matrix Utility Package           --
--               Harris Binding r3, 18 Aug 85              --
--                                                         --
--                Kevin E. Leinbach, GSO-85D               --
-------------------------------------------------------------
with GKS_CONFIGURATION; use GKS_CONFIGURATION;

generic
     type ELEMENT_TYPE is private;

package GKS_MATRIX_UTILITIES is

    -- The next type is not in the Harris binding but is
    -- required to prevent run-time errors on the Vax/Unix
    -- using the Verdix VADS.

    subtype INDEX is NATURAL range 0 ..
                                   MAX_SIZE_FOR_MATRIX;

    type MATRIX_OF is array ( POSITIVE range <>,
                              POSITIVE range <>)
                                       of ELEMENT_TYPE;

    type VARIABLE_MATRIX_OF ( DX : INDEX := 0;
                              DY : INDEX := 0) is
         record         -- INDEX replaces binding's NATURAL
            MATRIX: MATRIX_OF( 1..DX, 1..DY);
         end record;

end GKS_MATRIX_UTILITIES;
```

Appendix L:

Package EXTERNAL_TYPES

```
-------------------------------------------------------------
--                    GKS External Types                   --
--                Harris Binding r3, 18 Aug 85             --
--                                                         --
--                  Kevin E. Leinbach, GSO-85D             --
-------------------------------------------------------------
-- This package includes the rest of GKS's defined types
-- that are not included in BASIC_TYPES.
-- They appear in generally the same order as Ruegg had
-- them in his original EXTERNAL_TYPES package.

-- To save space here, the comments appearing in the binding
-- along with each type are not repeated here unless they
-- are different or the types are altered.

with basic_types;
with gks_configuration;
with gks_coordinate_system;
with gks_list_utilities;
with gks_matrix_utilities;

package EXTERNAL_TYPES is

    use basic_types;
    use gks_configuration;

    type CHOICE_VALUE is new POSITIVE;

    type VALUATOR_INPUT_VALUE is digits PRECISION;

    type PICK_ID is new POSITIVE;

    package PICK_IDS is new GKS_LIST_UTILITIES( PICK_ID);

    type SEGMENT_NAME is new POSITIVE;

    type WC_TYPE is digits PRECISION;

    type WS_ID is new POSITIVE;

    type ASF is ( BUNDLED, INDIVIDUAL);

    type ASF_LIST is
         record
            LINETYPE            : ASF;
            LINE_WIDTH          : ASF;
            LINE_COLOUR         : ASF;
```

```
            MARKER_TYPE          : ASF;
            MARKER_SIZE          : ASF;
            TEXT_FONT_PRECISION  : ASF;
            CHAR_EXPANSION       : ASF;
            CHAR_SPACING         : ASF;
            TEXT_COLOUR          : ASF;
            INTERIOR_STYLE       : ASF;
            STYLE_INDEX          : ASF;
            FILL_AREA_COLOUR     : ASF;
         end record;

      type ATTRIBUTES_FLAG is ( CURRENT, SPECIFIED);

      type ATTRIBUTES_USED_TYPE is ( POLYLINE_ATTRIBUTES,
                                     POLYMARKER_ATTRIBUTES,
                                     TEXT_ATTRIBUTES,
                                     FILL_AREA_ATTRIBUTES);

      package ATTRIBUTES_USED is new
              GKS_LIST_UTILITIES( ATTRIBUTES_USED_TYPE);

   -- type SCALE_FACTOR is digits PRECISION;

   -- This type is located in GKS_CONFIGURATION.  Its
   -- inclusion here causes an error in the next type.  A
   -- description of this error is in LRM 3.4(15).

      type CHAR_EXPANSION is new SCALE_FACTOR range
           SCALE_FACTOR'SAFE_SMALL
           ..SCALE_FACTOR'LAST;

      type RANGE_OF_EXPANSIONS is
           record
              MIN: CHAR_EXPANSION;
              MAX: CHAR_EXPANSION;
           end record;

      package WC is new GKS_COORDINATE_SYSTEM( WC_TYPE);

      type CHAR_SPACING is new SCALE_FACTOR;

   -- type CHOICE_DATA_RECORD( PROMPT_ECHO_TYPE :
   --                          CHOICE_PROMPT_ECHO_TYPE := 1)
   --      is private;

      type CHOICE_REQUEST_STATUS is
                   ( OK, NOCHOICE, NONE);

      subtype CHOICE_STATUS is CHOICE_REQUEST_STATUS range
                     OK .. NOCHOICE;

      package CHOICE_PROMPT_ECHO_TYPES is new
              GKS_LIST_UTILITIES( CHOICE_PROMPT_ECHO_TYPES);
```

```
type CLIPPING_INDICATOR is ( CLIP, NOCLIP);

type COLOUR_AVAILABLE is ( COLOUR, MONOCHROME);

type PIXEL_COLOUR_INDEX is new INTEGER range
     -1 .. INTEGER'LAST;

subtype COLOUR_INDEX is PIXEL_COLOUR_INDEX range
        0 .. PIXEL_COLOUR_INDEX'LAST;

package COLOUR_INDICES is new
        GKS_LIST_UTILITIES( COLOUR_INDEX);

package COLOUR_MATRICES is new
        GKS_MATRIX_UTILITIES( COLOUR_INDEX);

type INTENSITY is digits PRECISION range 0.0 .. 1.0;

type COLOUR_REPRESENTATION is
     record
        RED   : INTENSITY;
        GREEN : INTENSITY;
        BLUE  : INTENSITY;
     end record;

subtype CONNECTION_ID is STRING;

type VARIABLE_CONNECTION_ID( LENGTH: SUB_NATURAL := 0) is
     record
        CONNECT : CONNECTION_ID ( 1..LENGTH);
     end record;

-- SUB_NATURAL replaces the Harris binding's NATURAL.
-- SUB_NATURAL is defined in GKS_CONFIGURATION.

type CONTROL_FLAG is ( CONDITIONALLY, ALWAYS);

type DC_TYPE is digits PRECISION;

package DC is new
        GKS_COORDINATE_SYSTEM( DC_TYPE);

type DC_UNITS is ( METRES, OTHER);

type DEFERRAL_MODE is ( ASAP, BNIG, BNIL, ASTI);

type DEVICE_NUMBER is new POSITIVE;

type DISPLAY_CLASS is ( VECTOR_DISPLAY,
                        RASTER_DISPLAY,
                        OTHER_DISPLAY);

type DISPLAY_SURFACE_EMPTY is ( EMPTY, NOTEMPTY);
```

```
type DYNAMIC_MODIFICATION is ( IRG, IMM);

type ECHO_SWITCH is ( ECHO, NOECHO);

subtype ERROR_FILE_TYPE is STRING;

type ERROR_INDICATOR is new INTEGER;

type FILL_AREA_INDEX is new POSITIVE;

type INTERIOR_STYLE is ( HOLLOW, SOLID, PATTERN, HATCH);

type STYLE_INDEX is new INTEGER;

subtype PATTERN_INDEX is STYLE_INDEX range
        1 .. STYLE_INDEX'LAST;

subtype HATCH_STYLE is STYLE_INDEX;

type FILL_AREA_DATA (ATTRIBUTES : ATTRIBUTES_FLAG
                                := CURRENT) is
     record
        case ATTRIBUTES is
           when SPECIFIED =>
              STYLE_ASF        : ASF;
              STYLE_INDEX_ASF  : ASF;
              COLOUR_ASF       : ASF;
              INDEX            : FILL_AREA_INDEX;
              INTERIOR         : INTERIOR_STYLE;
              STYLE            : STYLE_INDEX;
              COLOUR           : COLOUR_INDEX;
           when CURRENT => null;
        end case;
     end record;

package FILL_AREA_INDICES is new
        GKS_LIST_UTILITIES( FILL_AREA_INDEX);

type GDP_ID is new INTEGER;

package GDP_IDS is new
        GKS_LIST_UTILITIES( GDP_ID);

type GKS_LEVEL is ( Lma, Lmb, Lmc,
                    L0a, L0b, L0c,
                    L1a, L1b, L1c,
                    L2a, L2b, L2c);

type GKSM_ITEM_TYPE is new NATURAL;

-- type GKSM_DATA_RECORD ( ITEM_TYPE : GKSM_ITEM_TYPE := 0)
--      is private;
```

```
        package HATCH_STYLES is new
            GKS_LIST_UTILITIES( HATCH_STYLE);

        type HORIZONTAL_ALIGNMENT is ( NORMAL, LEFT,
                                       CENTRE, RIGHT);

        type INPUT_CLASS is ( NONE, LOCATOR_INPUT, STROKE_INPUT,
                              VALUATOR_INPUT, CHOICE_INPUT,
                              PICK_INPUT, STRING_INPUT);

        subtype INPUT_QUEUE_CLASS is INPUT_CLASS range
                LOCATOR_INPUT .. STRING_INPUT;

        type INPUT_STATUS is ( OK, NONE);

        type INPUT_STRING( LENGTH : SUB_NATURAL := 0) is
             record
                CONTENTS: STRING( 1..LENGTH);
             end record;
        -- SUB_NATURAL replaces the Harris binding's NATURAL
        -- SUB_NATURAL is defined in GKS_CONFIGURATION

        package INTERIOR_STYLES is new
                GKS_LIST_UTILITIES( INTERIOR_STYLES);

        type INVALID_VALUES_INDICATOR is ( ABSENT, PRESENT);

        type LINETYPE is new INTEGER;

        type POLYLINE_INDEX is new POSITIVE;

        type LINE_WIDTH is new SCALE_FACTOR range
             0.0 .. SCALE_FACTOR'LAST;

        type LINEDATA( ATTRIBUTES: ATTRIBUTES_FLAG:= CURRENT) is
             record
                case ATTRIBUTES is
                   when SPECIFIED =>
                      LINE_ASF    : ASF;
                      WIDTH_ASF   : ASF;
                      COLOUR_ASF  : ASF;
                      INDEX       : POLYLINE_INDEX;
                      LINE        : LINETYPE;
                      WIDTH       : LINE_WIDTH;
                      COLOUR      : COLOUR_INDEX;
                   when CURRENT => null;
                end case;
            end record;

        package LINETYPES is new
                GKS_LIST_UTILITIES ( LINETYPE);

    -- type LOCATOR_DATA_RECORD( PROMPT_ECHO_TYPE :
```

```
--                                  LOCATOR_PROMPT_ECHO_TYPE := 1)
--       is private;

   package LOCATOR_PROMPT_ECHO_TYPES is new
           GKS_LIST_UTILITIES( LOCATOR_PROMPT_ECHO_TYPE);

   type MARKER_TYPE is new INTEGER;

   type POLYMARKER_INDEX is new POSITIVE;

   type MARKER_SIZE is new SCALE_FACTOR range
        0.0 .. SCALE_FACTOR'LAST;

   type MARKER_DATA( ATTRIBUTES: ATTRIBUTES_FLAG
                                 := CURRENT) is
        record
           case ATTRIBUTES is
              when SPECIFIED =>
                 MARKER_ASF : ASF;
                 SIZE_ASF   : ASF;
                 COLOUR_ASF : ASF;
                 INDEX      : POLYMARKER_INDEX;
                 MARKER     : MARKER_TYPE;
                 SIZE       : MARKER_SIZE;
                 COLOUR     : COLOUR_INDEX;
              when CURRENT => null;
           end case;
        end record;

   package MARKER_TYPES is new
           GKS_LIST_UTILITIES( MARKER_TYPE);

   type MEMORY_UNITS is range 0 .. MAX_MEMORY_UNITS;

   type MORE_EVENTS is ( NOMORE, MORE);

   type NDC_TYPE is digits PRECISION;

   package NDC is new
           GKS_COORDINATE_SYSTEM( NDC_TYPE);

   type NEW_FRAME_NECESSARY is ( NO, YES);

   type OPERATING_MODE is ( REQUEST_MODE,
                            SAMPLE_MODE,
                            EVENT_MODE);

   type OPERATING_STATE is ( GKCL, GKOP, WSOP, WSAC, SGOP);

   package PATTERN_INDICES is new
           GKS_LIST_UTILITIES( PATTERN_INDEX);

-- type PICK_DATA_RECORD( PROMPT_ECHO_TYPE :
```

```
--                              PICK_PROMPT_ECHO_TYPE := 1)
--       is private;

   type PICK_REQUEST_STATUS is ( OK, NOPICK, NONE);

   subtype PICK_STATUS is PICK_REQUEST_STATUS range
           OK .. NOPICK;

   package PICK_PROMPT_ECHO_TYPES is new
           GKS_LIST_UTILITIES( PICK_PROMPT_ECHO_TYPE);

   package PIXEL_COLOUR_MATRICES is new
           GKS_MATRIX_UTILITIES( PIXEL_COLOUR_INDEX);

   package POLYLINE_INDICES is new
           GKS_LIST_UTILITIES( POLYLINE_INDEX);

   package POLYMARKER_INDICES is new
           GKS_LIST_UTILITIES( POLYMARKER INDEX);

   type CHOICE_PROMPT is ( OFF, ON);

   package CHOICE_PROMPTS is new
           GKS_LIST_UTILITIES( CHOICE_PROMPT);

   type CHOICE_PROMPT_STRING( LENGTH: SUB_NATURAL:= 0) is
        record
           CONTENTS: STRING( 1..LENGTH);
        end record;
   -- SUB_NATURAL replaces the Harris binding's NATURAL
   -- SUB_NATURAL is defined in GKS_CONFIGURATION

   type CHOICE_PROMPT_STRING_ARRAY is array
           ( POSITIVE range <>) of CHOICE_PROMPT_STRING;

   type CHOICE_PROMPT_STRING_LIST( LENGTH: SUB_NATURAL:= 0)
     is record
           LIST: CHOICE_PROMPT_STRING_ARRAY( 1..LENGTH);
        end record;
   -- SUB_NATURAL replaces the HArris binding's NATURAL
   -- SUB_NATURAL is defined in GKS_CONFIGURATION

   type RADIANS is digits PRECISION;

   type RASTER_UNITS is new POSITIVE;

   type RASTER_UNIT_SIZE is
        record
           X : RASTER_UNITS;
           Y : RASTER_UNITS;
        end record;

   type UPDATE_REGENERATION_FLAG is ( PERFORM, POSTPONE);
```

```
      type REGENERATION_MODE is ( SUPPRESSED, ALLOWED);

      type RELATIVE_PRIORITY is ( HIGHER, LOWER);

      type RETURN_VALUE_TYPE is ( SET, REALIZED);

      subtype SECONDS is DURATION range 0.0 .. DURATION'LAST;

      type SEGMENT_DETECTABILITY is ( UNDETECTABLE,
                                      DETECTABLE);

      type SEGMENT_HIGHLIGHTING is ( NORMAL, HIGHLIGHTED);

      package SEGMENT_NAMES is new
              GKS_LIST_UTILITIES( SEGMENT_NAME);

      type SEGMENT_PRIORITY is digits PRECISION range 0.0..1.0;

      type SEGMENT_VISIBILITY is ( VISIBLE, INVISIBLE);

   -- type STRING_DATA_RECORD( PROMPT_ECHO_TYPE :
   --                          STRING_PROMPT_ECHO_TYPE := 1)
   --      is private;

      package STRING_PROMPT_ECHO_TYPES is new
              GKS_LIST_UTILITIES( STRING_PROMPT_ECHO_TYPE);

   -- type STROKE_DATA_RECORD( PROMPT_ECHO_TYPE :
   --                          STROKE_PROMPT_ECHO_TYPE := 1)
   --      is private;

       package STROKE_PROMPT_ECHO_TYPES is new
               GKS_LIST_UTILITIES( STROKE_PROMPT_ECHO_TYPE);

      subtype SUBPROGRAM_NAME is STRING;

      type VARIABLE_SUBPROGRAM_NAME( LENGTH: SUB_NATURAL:= 0)
        is record
              CONTENTS: SUBPROGRAM_NAME( 1..LENGTH);;
           end record;
      -- SUB_NATURAL replaces the Harris binding's NATURAL
      -- SUB_NATURAL is defined in GKS_CONFIGURATION

      type VERTICAL_ALIGNMENT is ( NORMAL, TOP, CAP,
                                   HALF, BASE, BOTTOM);

      type TEXT_ALIGNMENT is
           record
              HORIZONTAL : HORIZONTAL_ALIGNMENT;
              VERTICAL   : VERTICAL_ALIGNMENT;
           end record;
```

```
type TEXT_EXTENT_PARALLELOGRAM is
     record
        LOWER_LEFT  : WC.POINT;
        LOWER_RIGHT : WC.POINT;
        UPPER_LEFT  : WC.POINT;
        UPPER_RIGHT : WC.POINT;
     end record;

type TEXT_PRECISION is ( STRING_PRECISION,
                         CHAR_PRECISION,
                         STROKE_PRECISION);

type TEXT_FONT is new INTEGER;

type TEXT_FONT_PRECISION is
     record
        FONT      : TEXT_FONT;
        PRECISION : TEXT_PRECISION;
     end record;

package TEXT_FONT_PRECISIONS is new
        GKS_LIST_UTILITIES( TEXT_FONT_PRECISION);

type TEXT_INDEX is new POSITIVE;

package TEXT_INDICES is new
        GKS_LIST_UTILITIES( TEXT_INDEX);

type TEXT_PATH is ( RIGHT, LEFT, UP, DOWN);

type TRANSFORMATION_FACTOR is
     record
        X : NDC_TYPE;
        Y : NDC_TYPE;
     end record;

type TRANSFORMATION_MATRIX is array ( 1..2, 1..3) of
     NDC_TYPE;


type TRANSFORMATION_NUMBER is new NATURAL;

type TRANSFORMATION_PRIORITY_ARRAY is array
       ( POSITIVE range <>) of TRANSFORMATION_NUMBER;

type TRANSFORMATION_PRIORITY_LIST( LENGTH:
                                   SUB_NATURAL:= 0) is
     record
        CONTENTS:
           TRANSFORMATION_PRIORITY_ARRAY( 1..LENGTH);
     end record;
-- SUB_NATURAL replaces the Harris binding's NATURAL
-- SUB_NATURAL is defined in GKS_CONFIGURATION
```

```
      type UPDATE_STATE is ( NOTPENDING, PENDING);

   -- type VALUATOR_DATA_RECORD( PROMPT_ECHO_TYPE :
   --                            VALUATOR_PROMPT_ECHO_TYPE:= 1)
   --      is private;

      package VALUATOR_PROMPT_ECHO_TYPES is new
              GKS_LIST_UTILITIES( VALUATOR_PROMPT_ECHO_TYPE);

      type WS_CATEGORY is ( OUTPUT, INPUT, OUTIN,
                            WISS, MO, MI);

      package WS_IDS is new
              GKS_LIST_UTILITIES( WS_ID);

      type WS_STATE is ( INACTIVE, ACTIVE);

      type WS_TYPE is range 1..MAX_WS_TYPE;

      package WS_TYPES is new
              GKS_LIST_UTILITIES( WS_TYPE);

      -- The following types, given in the binding as private,
      -- were used by Ruegg as follows (with Harris r3 changes
      -- annotated:

      type LOCATOR_DATA_RECORD is
           record
              prompt_echo : locator_prompt_echo_type;
              control     : attributes_used_type;
              line        : linedata;
                            -- used to be 'line_data'  KEL
              fill        : fill_area_data;
           end record;

      type STROKE_DATA_RECORD is
           record
              buffer         : integer;
              edit_position  : integer;
              interval       : wc.point;
              time           : seconds;
              prompt_echo    : stroke_prompt_echo_type;
              marker         : marker_data;
              line           : linedata;
                               -- used to be 'line_data'  KEL
           end record;


      type VALUATOR_DATA_RECORD is
           record
              prompt_echo : valuator_prompt_echo_type;
              low_value   : valuator_input_value;
                            -- used to be 'input_value'  KEL
```

```
            high_value  : valuator_input_value;
                          -- used to be 'input_value'  KEL
        end record;

    type CHOICE_DATA_RECORD is
        record
            prompt_echo : choice_prompt_echo_type;
            choices     : integer;
            the_prompts : choice_prompts.list_of;
                          -- used to be 'prompts.list_of' KEL
            strings     : choice_prompt_string_list;
                          -- used to be 'prompt_strings.
                          -- list_of'  KEL
            segment     : segment_name;
            pick        : pick_ids.list_of;
        end record;

    type PICK_DATA_RECORD is
        record
            prompt_echo : pick_prompt_echo_type;
        end record;

    type STRING_DATA_RECORD is
        record
            prompt_echo     : string_prompt_echo_type;
            buffer          : integer;
            cursor_position : integer;
        end record;

 -- private

       -- do them all later

end EXTERNAL_TYPES;
```

Appendix M:

Points of Contact

The following information is provided to offer sources of information for the various aspects of GKS implemented in the Ada language.

AMERICAN NATIONAL STANDARD GKS:

American National Standards Institute
1430 Broadway
New York, NY 10018 (212)-354-3300

Mr. Peter Bono
Chairman, ANSI Committee X3H3
Graphics Software (Connecticut) (203)-464-9350/2165

GKS TESTING AND EVALUATION:

Mr. Steve Carson
Chairman, ANSI/ISO Certification Group
GSC Associates
Hawthorne, CA (213)-978-9351

Mr. Mark Skall
Manager, Data Management Systems Group
National Bureau of Standards
Room A-265, Technology Building
Gaithersburg, MD 20899 (301)-921-2431

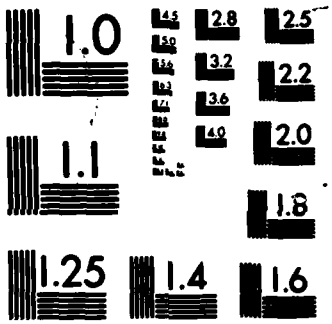
ADA BINDING:

Ms. Geraldine Cuthbert
Ada Binding Team Chief
Harris Corporation
505 John Rodes Boulevard
Melbourne, FL 32901 (305)-242-5079

Ms. Barbara Furtney
Ada Binding Team Member
Harris Corporation
505 John Rodes Boulevard
Melbourne, FL 32901 (305)-242-5324

AD-A163 036 PORTABILITY EVALUATION OF AFIT-GKS AN ADA IMPLEMENTATION OF THE GRAPHICAL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/2

UNCLASSIFIED K E LEINBACH DEC 85 AFIT/GSO/MA/85D-4 F/G 9/2 NL



DOCUMENTS:

Ms. Catherine A. Kachurik
Secretariat: Computer and Business Equipment
Manufacturers Association (CBEMA)
311 First Street NW
Suite 500
Washington, DC 20001 (202)-737-8888

VERDIX ADA DEVELOPMENT SYSTEM:

Dr. Seifert
Ada Product Support Office
VERDIX Corporation (703)-378-7600

VERDIX "arpanet" address: vrdxhq!apso@seismo

Bibliography

American National Standards Institute. Draft: American National Standard Graphical Kernel System. New York: American National Standards Institute, 1984.

American National Standards Institute. Draft: American National Standard Computer Graphics Interface. New York: American National Standards Institute, 1985.

Barnes, J.G.P Programming in Ada. London: Addison-Wesley, 1982.

Booch, Grady. Software Engineering with Ada. Menlo Park, California: Benjamin/Cummings, 1983.

Brodlie, K.W. "GKS Certification -- An Overview," Computing & Graphics, 8: 13-17 (Number 1, 1984).

Cuthbert, Geraldine, GKS Binding Team Chief. Telephone interview. Harris Corp., Melbourne, FL, 30 Jul 1985.

Department of Defense. Military Standard Ada Programming Language. ANSI/MIL-STD-1815A. Washington, D.C.: Ada Joint Programming Office (January 1983).

Enderle, G. and others. Computer Graphics Programming. Berlin: Springer-Verlag, 1984.

Furtney, Barbara, GKS Binding Team Member. Telephone interview. Harris Corp., Melbourne, FL, 17 Oct 1985a.

-----. Telephone interview. Harris Corp., Melbourne, FL, 18 Aug 1985b.

Hopgood, F.R.A. and others. Introduction to the Graphical Kernel System (GKS). New York: Academic Press, 1983.

Munro, Allen. "The Simplicity of Modularity," Softalk, 3: 209-210+ (March 1983).

National Bureau of Standards. GKS Validation Routines Documentation. Gaithersburg, MD: National Bureau Of Standards, 1985.

Raster Technologies. Raster Technologies Model One/25-S Programming Guide. Revision 1.0. Raster Technologies, Inc., 16 Feb 1984.

-----. Raster Technologies Model One Display List Firmware Programming Guide. Revision 1.0. Raster Technologies, Inc., 17 May 1983a.

-----. Raster Technologies Model One/25 Programming Guide. Revision 4.1. Raster Technologies, Inc., 12 Dec 1983b.

Ruegg, 2Lt Raymond S. AFIT_GKS -- a GKS Implementation in the Ada Programming Language. MS thesis, AFIT/GCS/MA/-84D-5. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1984.

VAda-040-03-0101. VERDIX Ada Development System Operations Manual. Version V4.06. VERDIX Corp, Mar 1985.

X3H3/83-95r1. Draft GKS Binding to ANSI Ada. Harris Corporation, Government Information Systems Software Operation, Melbourne, FL, 20 Jul 1984. Contract F49642-83-C0083.

X3H3/83-95r2. Draft GKS Binding to ANSI Ada. Harris Corporation, Government Information Systems Software Operation, Melbourne, FL, 1 Feb 1985. Contract F49642-84-C-0176.

Vita

Captain Kevin Edward Leinbach was born on 15 August 1953 in Reading, Pennsylvania. He graduated from Exeter Township High School in 1971 and attended Wentworth Military Academy and Junior College in Lexington, Missouri, for one year. He then entered the United States Air Force Academy and graduated in 1976 with a Bachelor of Science in Astronautical Engineering. He completed Undergraduate Pilot Training at Columbus AFB, Mississippi, in November 1977, and then entered F-4 upgrade training at MacDill AFB, Florida. He then flew with the F-4G Advanced Wild Weasel squadrons at Spangdahlem AB, West Germany, and George AFB, California, until entering the School of Engineering, Air Force Institute of Technology, in June 1984.

Permanent address: 1530 Fairview Street
Reading, PA 19606

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A163 836

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GSO/MA/85D-4	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Kevin E. Leinbach, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1985 December; 15. PAGE COUNT: 101	
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES — FIELD: 09; GROUP: 02; SUB. GR.:	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Computer Graphics, Debugging (Computers), Compilers, Computer Program Reliability

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: PORTABILITY EVALUATION OF AFIT_GKS: AN ADA IMPLEMENTATION of the GRAPHICAL KERNEL SYSTEM

Thesis Chairman: Richard R. Gross, Lt Col, USAF
Assistant Professor of Mathematics and Computer Science

Approved for public release IAW AFR 190-1.
LYNN E. WOLAVER - 16 JAN 86
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Richard R. Gross, Lt Col, USAF	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3098; 22c. OFFICE SYMBOL: AFIT/ENC

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

The purposes of this study were to develop an Ada implementation of the American National Standard Institute's new standard for computer graphics, the Graphical Kernel System (GKS), and to have it operating on an Air Force Institute of Technology (AFIT) computer system. The basis for this implementation was AFIT_GKS, a subset of GKS designed and coded by 2Lt R. Scott Ruegg on the Aeronautical System Division's Rolm Data General Computer for his 1984 AFIT thesis, AFIT GKS -- A GKS Implementation in the Ada Programming Language. It seemed possible to install AFIT_GKS on an AFIT computer, to improve AFIT_GKS by modifying it to address more capable display terminals, and to initiate testing and evaluation of AFIT_GKS in accordance with the methods and guidelines currently under study by the National Bureau of Standards.

Considerable difficulty in porting AFIT_GKS to AFIT's computer was encountered. Some of these difficulties were caused by problems with AFIT's compiler itself, and others by the substantial changes to the Ada binding to GKS. As a result, no device interfacing or evaluation work could be accomplished. However, a framework for development of GKS in Ada at AFIT was implemented, debugged, tested, and documented, and this framework can serve well as the foundation for continued work on this subject.

END

FILMED

3-86

DTIC